ELEX MEDIA KOMPUTINDO

SERI APLIKASI PEMROGRAMAN

PEMROGRAMAN ANIMASI DAN GAME PROFESIONAL 4

Disertai disket yang berisi contoh-contoh program

AGUSTINUS NALWAN

SERI APLIKASI PEMROGRAMAN

PEMROGRAMAN ANIMASI DAN GAME PROFESIONAL 4

AGUSTINUS NALWAN

Seri Aplikasi Pemrograman

Pemrograman Animasi dan Game Profesional 4

Sanksi Pelanggaran Pasal 44:
Undang-undang Nomor 12 Tahun 1997 Tentang
Perubahan atas Undang-undang Nomor 6 Tahun 1982 Tentang Hak Cipta
Sebagaimana Telah Diubah dengan Undang-undang Nomor 7 Tahun 1987

1. Barangsiapa dengan sengaja dan tanpa hak mengumumkan atau memperbanyak suatu ciptaan atau memberi izin untuk itu, dipidana dengan pidana penjara paling lama 7 (tujuh) tahun dan/atau denda paling banyak Rp. 100.000.000,- (seratus juta rupiah).

2. Barangsiapa dengan sengaja menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran Hak Cipta sebagaimana dimaksud dalam ayat (1), dipidana dengan pidana penjara paling lama 5 (lima) tahun dan/atau denda paling banyak Rp. 50.000.000,- (lima puluh juta rupiah).

Seri Aplikasi Pemrograman

Pemrograman Animasi dan Game Profesional 4

Agustinus Nalwan

Penerbit PT Elex Media Komputindo
Kelompok Gramedia - Jakarta

Seri Aplikasi Pemrograman
Pemrograman Animasi dan Game Profesional 4
Agustinus Nalwan
© 1998, PT Elex Media Komputindo, Jakarta
Hak cipta dilindungi undang-undang
Diterbitkan pertama kali oleh
PT Elex Media Komputindo
Kelompok Gramedia, Anggota IKAPI, Jakarta 1998

12198074
ISBN 979-20-0504-8

Dilarang keras menerjemahkan, memfotokopi, atau memperbanyak sebagian atau seluruh isi buku ini tanpa izin tertulis dari Penerbit.

Dicetak oleh Percetakan PT Gramedia, Jakarta
Isi di luar tanggung jawab percetakan

DAFTAR ISI

BAB 1 BERPINDAH DARI TURBO PASCAL KE C

1.1 BODI PROGRAM 1
Definisi Library pada Main Program 1
Format dari Bodi Program 2
1.2 STRUKTUR DATA 3
Variabel-variabel Standard 3
Konstanta 5
Variabel String 6
Array 8
Record 9
1.3 OPERASI VARIABEL 10
Operasi Aritmatik 10
Konversi Variabel (Casting) 11
1.4 FUNGSI SELEKSI 11
If-Then-Else 11
Case 13
1.5 PERULANGAN 13
For Loop 13
While do loop 15
1.6 PEMANGGILAN INTERRUPT DAN INPUT OUTPUT 15
Interrupt 15
Menampilkan Tulisan ke Layar 16
Membaca Input melalui keyboard 19
1.7 PENGGUNAAN SUBROUTINE 21
Pembuatan Procedure 21
Pembuatan Function 22
Mengirimkan Parameter ke Subroutine 23
Prototype dari Subroutine 28
1.8 FILE 29
Text File 29
Binary File 31
1.9 PEMBUATAN UNIT 33
Cara memanggil unit 33
Pembuatan Unit dalam C 34

1.10 PENGGUNAAN POINTER 35
Mengisi Alamat Memory dengan Data Lewat Pointer 36
Menduplikasi Data dalam Alamat Memory ke Alamat Memory yang Lain Lewat Pointer 37
Menduplikasi Data dari Variabel ke Alamat Memory Lewat Pointer 37
Kelebihan Pointer di C 38

BAB 2 PEMROGRAMAN DENGAN WATCOM C

2.1 PENGENALAN WATCOM C DAN PROTECTED MODE 40
2.2 INSTALASI WATCOM C 44
2.3 PENGGUNAAN UTILITY WATCOM C 45
WPP386.EXE Compiler 45
WLINK.EXE Linker 46
WD.EXE Debugger 47
WHELP.EXE Help Utility 48
Program Pertama Kita di Watcom C 49
Pengaturan Window 51
Pencarian String 52
Pelacakan Jalannya Program 53
Melihat Harga variabel 56
Melihat Harga Variabel Global 57
Lain-lain 59

BAB 3 PENDAHULUAN PEMBAHASAN

3.1 UTILITY PEMBANTU 60
3.2 LISTING PROGRAM DAN EXECUTABLE FILE 62
3.3 HAL-HAL YANG PERLU DIPERHATIKAN 62

BAB 4 DASAR PEMROGRAMAN SUPER VGA

4.1 MENELITI CARD SUPER VGA 65
4.2 MENGAKSES CARD SUPER VGA 67
4.3 ALOKASI TITIK MENGGUNAKAN TABEL WARNA 70
4.4 ALOKASI TITIK TANPA TABEL WARNA 71
4.5 PENGGUNAAN LAYAR BUATAN (VIRTUAL SCREEN) 72

BAB 5 PERAKITAN GRAPHICS LIBRARY 8 BIT PERTAMA

5.1 PEMROGRAMAN ASSEMBLY DENGAN REGISTER 32 BIT 75
5.2 SISTEM VIEWPORT 77
5.3 FUNGSI-FUNGSI STANDAR 78
5.4 FUNGSI-FUNGSI PENAMPILAN BANGUN GEOMETRI 95
5.5 FUNGSI-FUNGSI PENAMPILAN SPRITE 148
5.6 TEKNIK EFEK PENCAHAYAAN DAN PEMBAYANGAN GAMBAR.....180

BAB 6 PERAKITAN GRAPHICS LIBRARY 8 BIT KEDUA

6.1 FUNGSI PARALLAX SCROLLING.....237
6.2 FUNGSI PENAMPILAN TULISAN PADA MODE GRAPHIC 8 BIT 255

BAB 7 PEMROGRAMAN SUPER VGA 16, 24, 32 BIT

7.1 GRAPHIC LIBRARY 24BIT.ASM 262
7.2 FUNGSI-FUNGSI STANDAR 24BIT.ASM.....268
7.3 PENAMPILAN GAMBAR 8 BIT KE MODE 24 BIT.....283
7.4 FUNGSI PENAMPILAN SPRITE STANDAR 289
7.5 EFEK GAMMA PADA SPRITE.....296
7.6 EFEK MULTI CHANNEL TRANSPARANSI PADA SPRITE.....300
7.7 EFEK SPRITE CAHAYA 306
7.8 FUNGSI-FUNGSI TAMBAHAN 311

PENDAHULUAN

Halo rekan-rekan pembaca. Kita berjumpa lagi dalam seri Pemrograman Animasi dan Game Profesional 4. Seperti janji-janji saya terdahulu untuk membahas mengenai Watcom C, DOS4GW, pemrograman Super VGA dan pemrograman 32 bit protected mode. Di buku ini kita akan membahas seluruh topik tersebut.

Bagi para pembaca yang belum pernah memprogram di C, jangan khawatir. Untuk berpindah dari Turbo Pascal ke C hanya diperlukan beberapa jam. Untuk itu pada Bab 1 akan disediakan tutorial singkat tentang bagaimana berpindah dari Turbo Pascal ke C secara cepat disertai dengan contoh-contoh program dalam versi Turbo Pascal beserta C-nya sehingga kita dapat membandingkan perbedaan dan kesamaannya. Tutorial ini akan membahas hampir dari seluruh fungsi C yang akan kita gunakan dalam buku ini.

Selanjutnya akan dijelaskan bagaimana cara instalasi dan penggunaan dari utility Watcom C. Hal ini termasuk bagaimana cara mengcompile program, cara men-debug program dengan debugger Watcom C yang super canggih.

Kedua materi di atas yaitu tentang penggunaan Watcom C dan berpindah dari Turbo Pascal ke C hanya merupakan sebagian kecil dibandingkan dengan materi pokok yang akan kita bahas di buku ini. Materi pokok ini ialah pemrograman Super VGA dan 32 bit protected mode yang akan membawa kita ke dunia pemrograman graphic resolusi tinggi hingga 1280 x 1024. Akan diajarkan bagaimana menampilkan gambar dan animasi pada mode graphic 8 bit, 16, 24 dan juga 32 bit yang memungkinkan kita menampilkan gambar dengan kualitas seperti apa yang Anda lihat pada software Adobe Photo Styler. Seiring dengan pembahasan mode graphic tersebut akan dibahas pula teknik-teknik dan efek-efek yang hanya dapat diciptakan dengan mode graphic di atas 8 bit seperti teknik pencahayaan, multi channel transparansi antara dua gambar, pemberian bayangan dan juga efek gamma correction yang ada pada software Adobe Photo Styler.

Ingin mengetahui lebih lanjut? Teruskan membaca.

Penyusun

BAB 1
BERPINDAH DARI TURBO PASCAL KE C

1.1 BODI PROGRAM

Dalam pemrograman C, yang harus selalu kita ingat ialah C sangat sensitif terhadap huruf besar dan huruf kecil (case sensitive). Contohnya jika kita hendak menggunakan for loop, for harus ditulis dalam huruf kecil. "For" akan diterima bagi C sebagai error. Memang untuk pertama bagi Anda yang sudah terbiasa dalam Turbo Pascal akan sedikit merasa kesulitan untuk beradaptasi, tetapi dengan sering berlatih hal ini bukanlah sesuatu yang sulit.

Definisi Library pada Main Program

Setiap kali kita membuat program pada Turbo Pascal, kita selalu mendefinisikan unit-unit atau library yang akan kita pergunakan. Contohnya Dos, Crt dan Graph. Pada C, kita juga harus melakukan hal yang sama.

Listing Program dalam Turbo Pascal

```
Uses GraphLib,DosLib;
.....
```

Listing Program dalam C

```
#include <GraphLib.h>
#include <DosLib.h>
.....
```

Listing program dalam Turbo Pascal di atas mendefinisikan library yang bernama GraphLib.TPU dan DosLib.TPU. Sedangkan program dalam C di atas mendefinisikan library yang bernama GraphLib.h dan DosLib.h.

Library yang selalu kita pergunakan dalam Turbo Pascal ialah Crt. Dalam C, library yang selalu kita gunakan ialah stdio.h. Library ini mengatur input output dari seluruh proses seperti transfer harga antarvariabel dan lain sebagainya.

Catatan

- Dalam C, Kita tidak dapat mendefinisikan beberapa library sekaligus dalam satu baris dengan menggunakan tanda koma seperti halnya pada Turbo Pascal.
- Format #include <nama library> di atas hanya berlaku untuk unit/library yang telah disediakan oleh C, untuk mendefinisikan library yang kita buat sendiri kita harus menggunakan format lain yang akan kita bahas pada bagian Pembuatan Unit.
- stdio.h harus selalu didefinisikan karena library ini selalu kita perlukan.

Format dari Bodi Program

Listing Program dalam Turbo Pascal

```
Uses GraphLib;              { definisi library   }
Var  A:Integer;             { deklarasi variabel }
Begin                       { main program       }
End.
```

Listing Program dalam C

```
#include <GraphLib.h>      { definisi library   }
int a;                     { deklarasi variabel }
void main()                { main program       }
{
}
```

Yang perlu diingat:

- Dalam C, main program merupakan procedure dan harus diberi nama main.
- Karena main program adalah procedure maka main program dapat menerima parameter. Hal ini digunakan sebagai pengganti fungsi ParamStr dari Turbo Pascal. Untuk lebih jelasnya akan kita bahas pada bagian pembuatan subroutine.
- Untuk membatasi proses, sebagai pengganti dari Begin dan End, dalam C digunakan "{" dan "}".
- Deklarasi variabel dalam C tidak perlu diawali dengan "Var".

Catatan

Dalam C, komentar harus disisipkan di antara tanda "/*" dan "*/" jika komentar tersebut menempati lebih dari satu baris. Jika komentar tersebut hanya sebesar satu baris maka cukup dilakukan tanda "//" di depan komentar tersebut.

Contoh Program dalam C

```
#include <stdio.h>
/* Komentar lebih dari satu
   baris harus dibatasi dengan
   tanda ini
*/
// Komentar yang hanya menempati satu baris
void main()
{
}
```

1.2 Struktur Data

Variabel-variabel Standard

Jenis-jenis variabel pada C jauh lebih beragam dibandingkan dengan Turbo Pascal. Jenis-jenis variabel tersebut dapat Anda lihat pada Tabel 1.1 berikut ini.

Tabel 1.1 Jenis-jenis Variabel dalam C dan Turbo Pascal

Jenis variabel (Pascal)	Ukuran	Limit	Jenis Variabel (C)
Char	1 byte	-128 hingga 127	char
Shortint	1 byte	-128 hingga 128	signed char
Byte	1 byte	0 hingga 255	unsigned char
Integer	2 byte	-32768 hingga 32767	int
integer	2 byte	-32768 hingga 32767	signed int
word	2 byte	0 hingga 65535	unsigned int
integer	2 byte	-32768 hingga 32767	short int
integer	2 byte	-32768 hingga 32767	signed short int
word	2 byte	0 hingga 65535	unsigned short int
longint	4 byte	-2147483648 hingga 2147483647	long int
longint	4 byte	-2147483648 hingga 2147483647	signed long int
-	4 byte	0 hingga 4294967295	unsigned long int
single	4 byte	3.4E-38 hingga 3.4E+38	float
Double	8 byte	1.7E-308 hingga 1.7E+308	long float
Double	8 byte	1.7E-308 hingga 1.7E+308	double
-	10 byte	3.4E-4932 hingga 1.1E+4932	long double

Listing Program dalam Turbo Pascal

```
Uses Crt,Dos;

Var A,B:Integer;
    C  :Real;

Begin
    A:=10;
    C:=4.5;
End.
```

Listing Program dalam C

```
#include <stdio.h>

int   a=10,b;
float c=4.5;

void main()
{
}
```

Yang perlu diingat:

- Dalam C dikenal istilah unsigned yang berarti variabel tidak mampu menampung bilangan negatif dan signed yang berarti variabel yang mampu menampung bilangan negatif.
- Dalam C dikenal istilah long yang berarti variabel tersebut lebih besar dari ukuran sebenarnya. Misalnya long int berarti lebih besar dari int. Pada compiler-compiler C tertentu seperti Borland C, untuk mendeklarasikan "long int" cukup dengan "long".
- Beberapa compiler C seperti Watcom C dan DJGPP ukuran dari jenis variabel di atas berbeda, untuk itu kita harus selalu berpedoman pada dokumentasi dari compiler yang bersangkutan.
- Char dan Byte tidak dibedakan dalam C. Keduanya digolongkan sebagai Char.
- Dalam C, kita dapat memberi harga pada suatu variabel langsung pada saat variabel tersebut dideklarasikan.
- Untuk memasukkan harga ke dalam variabel, dalam C cukup hanya menggunakan tanda "=". Misalnya A=5.
- Char selalu direpresentasikan dalam tanda kutip '.
  Misalnya: char a='h';

Konstanta

Konstanta dideklarasikan dengan menggunakan fungsi #define.

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Const Max = 100;
      Min = 5;
```

```
Var X,Y:Integer;

Begin
End.
```

Listing Program dalam C

```
#include <stdio.h>

#define Max 100
#define Min 5

int X,Y;

void main()
{
}
```

Variabel String

Variabel string-nya mana?

Dalam C tidak dikenal variabel string. Variabel string dibentuk dari array of Char. Dan yang paling penting untuk diingat ialah, string dalam C dibatasi dengan tanda ". Dalam Turbo Pascal string dibatasi dengan tanda '.

Deklarasi String dalam C	Deklarasi String dalam Turbo Pascal
Char H[50];	Var H:String[50];

Dalam Turbo Pascal, variabel kita dapat menganggap variabel String seperti halnya variabel biasa. Kita dapat melakukan operasi-operasi seperti berikut:

- if H='test' then
- H:=H+'tambahan';
- H:='halo !';

Kabar buruk.... Dalam C operasi-operasi seperti di atas tidak dapat dilakukan, seluruh operasi yang akan dilakukan pada variabel string harus melalui pemanggilan fungsi-fungsi yang telah disediakan oleh C.

Fungsi strcpy

Fungsi ini digunakan untuk memasukkan teks ke dalam variabel string.

Contoh:

Dalam C	Dalam Turbo Pascal
strcpy(H,"halo");	H:='halo';

Fungsi strcmp

Fungsi ini digunakan untuk membandingkan dua variabel string.

Contoh:

Dalam C	Dalam Turbo Pascal
if (strcmp(VarStr1,VarStr2)==0)	if VarStr1=VarStr2

Jika fungsi ini menghasilkan angka 0 berarti VarStr1 sama dengan VarStr2.

Jika fungsi ini menghasilkan angka -1 berarti VarStr1 lebih kecil dari VarStr2.

Jika fungsi ini menghasilkan angka 1 berarti VarStr1 lebih besar dari VarStr2.

Maksud dari lebih besar dan lebih kecil di sini ialah urutan dari teks tersebut secara alfabet.

Dengan ini kita dapat menggantikan fungsi > dan < dari Turbo Pascal untuk mengurutkan string.

Fungsi strcat

Fungsi ini digunakan untuk menambahkan teks pada akhir variabel string.

Contoh:

Dalam C	Dalam Turbo Pascal
strcat(H,"tambahan);	H:=H+'Tambahan';

Selain itu, fungsi ini juga dapat menambahkan sebuah variabel string pada akhir variabel string.

Contoh:

Dalam C	Dalam Turbo Pascal
Strcat(H,VarStr2);	H:=H+VarStr2;

Yang perlu diingat:

- Pada saat menambahkan teks pada sebuah variabel string misalnya VarStr1, hasil dari penambahan tersebut tidak boleh melebihi ukuran dari VarStr1 pada saat dideklarasikan, karena teks yang melebihi ukuran VarStr1 itu akan disimpan bertabrakan dengan variabel-variabel lain dalam memory.

Array

Cara deklarasi variabel array dalam C agak lain dengan deklarasi pada Turbo Pascal. Dalam C, pada saat mendeklarasikan array, kita harus memberitahukan ukuran array tersebut. Sedangkan limit array tersebut adalah mulai 0 hingga ukuran-1.

Deklarasi array dalam C	Deklarasi array dalam Turbo Pascal
int Test[100];	Var Test:Array[0..99] of Integer;
int Test[10][100];	Var Test:Array[0..9,0..99] of Integer;
char Test[50];	Var Test:Array[0..49] of Char;

Cara penggunaan array dalam C dan Pascal berbeda hanya pada penempatan index dari array berukuran lebih besar dari satu dimensi. Pada C digunakan kurung tambahan, sedangkan pada Turbo Pascal cukup dengan menggunakan tanda koma.

Penggunaan array dalam C	Penggunaan array dalam Turbo Pascal
Test[40]=5;	Test[40]:=5;
Test[4][5]=77;	Test[4,5]:=77;

Yang perlu diingat:

- Lain dengan Turbo Pascal, dalam C, penggunaan array yang melebihi batas tidak diberi peringatan oleh C compiler. Misalnya:

```
char Test[50];  // Array dideklarasikan berukuran 50.
Test[100]=4;    // Penggunaan array melebihi batas.
```

Record

Listing program dalam Turbo Pascal.

```
Uses Crt,Dos;

Type RMsiswa = Record
        Nama    : String[20];
       Alamat : String[30];
       Nilai   : Integer;
     End;

Var Mahasiswa:RMsiswa;

Begin
  Mahasiswa.Nama:='Oliver';
  Mahasiswa.Alamat:='Jl Jambu 13';
  Mahasiswa.Nilai:=40;
End.
```

Listing Program dalam C

```
#include <stdio.h>

struct RMsiswa
{
        char   Nama[20];
         char  Alamat[30];
         int    Nilai;
}

struct RMsiswa      Mahasiswa;

void main()
{
  strcpy(Mahasiswa.Nama,"Oliver");
  strcpy(Mahasiswa.Alamat,"Jl Jambu 13");
  Mahasiswa.Nilai=40;
}
```

1.3 Operasi Variabel

Operasi Aritmatik

Operasi aritmatik dalam C lebih banyak dibandingkan dengan Turbo Pascal. Lain dengan Turbo Pascal, operasi aritmatik tertentu seperti pembagian dengan tanda "/" dapat dilakukan pada variabel integer. (Pada Turbo Pascal kita harus menggunakan div untuk variabel integer.)

Oleh karena itu, untuk menghindari kesalahan compiler C menganggap suatu operasi aritmatik sebagai operasi integer, kita harus selalu mengikutsertakan angka di belakang koma pada setiap angka dalam melakukan operasi pecahan.

Misalnya:

```
float a=5/3;
```

Karena tidak ada angka di belakang koma, maka operasi ini dianggap sebagai operasi integer. Sehingga variabel a akan berisi 1.

Cara yang benar ialah:

```
float a=5.0/3.0;
```

Tabel 1.2 Operasi Aritmatik

Dalam C	Dalam Turbo Pascal
a=a+5;	a:=a+5;
a+=3;	a:=a+3;
a-=5;	a:=a-5;
a++;	inc(a);
a--;	dec(a);
a*=4;	a:=a*4;
a/=4;	a:=a/4;
a=a*4	a:=a*4;
a=a/10	a:=a/10;
>>	Shr
<<	Shl

Konversi Variabel (Casting)

Fasilitas ini diberikan oleh C compiler untuk mengatasi kesalahan dalam mengartikan operasi aritmatik seperti yang telah kita bahas di atas dengan contoh operasi 5/3 menghasilkan bilangan bulat 1.

Memang dengan cara mengikutsertakan angka di belakang desimal dapat memecahkan masalah tersebut. Tetapi bagaimana kalau operasi yang kita lakukan adalah operasi terhadap variabel misalnya:

```
int a=5,b=3;
float a=b/c;
```

Kita tentu saja tidak dapat menambahkan angka di belakang koma pada variabel b dan c. Untuk itu digunakan variabel casting sebagai berikut:

```
int a=5,b=3;
float a=(float)(b/c);
```

Dengan ini kita memberitahukan C compiler bahwa operasi di dalam tanda kurung tersebut ialah operasi bilangan pecahan (float).

Jenis variabel casting ini sama persis dengan jenis-jenis variabel yang ada di C seperti yang telah kita bahas pada bagian Data Structure.

Yang perlu diingat:

- Dalam C, kita dapat memindahkan harga antara dua variabel yang berbeda jenis misalnya:

```
float a=3.2;
int b=a;          // variabel b akan berisi 3 (pembulatan ke bawah dari 3.2).
```

1.4 Fungsi Seleksi

If-Then-Else

Seperti pada Turbo Pascal, C juga mengenal fungsi seleksi if then.

Listing Program dalam Turbo Pascal

```
Begin
  a:=5;
  if a=5 then
```

```
    Begin
      b:=1;
    End Else
    Begin
      b:=0;
    End;
End.
```

Listing Program dalam C

```
void main()
{
  a=5;
  if (a==5)
  {
    b=1;
  } else
  {
    b=0;
  }
}
```

Tabel 1.3 Operasi Pembanding dalam C dan Turbo Pascal

Operasi Pembanding dalam C	Operasi Pembanding dalam Turbo Pascal
||	or
&&	and
!	not
1	true
0	false
<	<
>	>
<=	<=
>=	>=
==	=

Yang perlu diingat:

- Tanda == (dua kali tanda sama dengan) digunakan sebagai pembanding antara dua harga dalam C.
- Dalam C pada setiap if, kondisi harus selalu diletakkan dalam tanda kurung.

Case

Dalam C, perintah yang sama dengan case dalam Turbo Pascal ialah switch.

Listing Program dalam Turbo Pascal

```
case a of
  1:Begin
      b:=1;
    End;
  2:Begin
      b:=2;
    End;
  ELSE b:=3;
End;
```

Listing Program dalam C

```
switch (a)
{
  case 1:
       b=1;
       break;
  case 2:
       b=2;
       break;
  default:
       b=3;
}
```

1.5 Perulangan

For Loop

Penggunaan for loop dalam C jauh lebih bebas dari Turbo Pascal. Format dari For loop dalam C adalah sebagai berikut:

```
for (perintah1; kondisi; perintah2)
{
}
```

Pada perintah1 dapat dimasukkan proses yang hanya dilakukan sebelum dilakukan loop.

Sedangkan pada kondisi harus diisikan dengan kondisi di mana loop masih terus berlangsung.

Pada perintah2 dapat dimasukkan proses yang akan dilakukan setiap kali satu loop berlangsung.

Listing Program dalam Turbo Pascal

```
for a:=1 to 10 do
  Begin
  End;
```

Listing Program dalam C

```
for (a=1;a<=10;a++)
{
}
```

Perintah1 pada for loop di atas memberi harga a=1 sebelum loop berlangsung.

Di sini kondisi menyatakan bahwa loop selama harga a masih lebih kecil atau sama dengan 10.

Perintah2 pada for loop di atas menambah harga a setiap kali satu loop berlangsung.

Beberapa variasi penggunaan for loop dapat dilihat pada contoh-contoh berikut:

Listing Program dalam C

```
b=40;
for (a=1;(a<=100||b==5);a+=3)
{
  b=b-3;
}

for (a=100;a>=10;a--)
{
}
```

While do loop

Fungsi loop terakhir ialah While do (C tidak memiliki fungsi repeat until). While do dapat dilakukan dengan dua cara yaitu yang pertama:

```
While (kondisi)
{
}
```

Kondisi ialah kondisi di mana loop masih terus berlangsung.

Cara yang kedua hampir sama dengan repeat until yaitu:

```
do
{
} while (kondisi);
```

Kondisi ialah kondisi di mana loop masih terus berlangsung.

1.6 Pemanggilan Interrupt dan Input Output

Interrupt

Dalam Turbo Pascal, fungsi interrupt terdapat pada unit Dos yang telah disediakan. Dalam C, fungsi interrupt disediakan dalam library dos.h. Untuk itu sebelum kita hendak menggunakan interrupt, kita harus mendefinisikan library dos.h pada awal program. Seperti halnya pada Turbo Pascal, dalam C register disimpan di dalam record/ structure yaitu bernama REGPACK.

Listing Program dalam Turbo Pascal

```
Uses Crt,Dos;

Var Reg       : Registers;

Begin
  Reg.ax:=10;
  Reg.bx:=20;
  Intr($10,Reg);
End.
```

Listing Program dalam C

```
#include <stdio.h>
#include <dos.h>
```

```
struct REGPACK reg;

void main()
{
  reg.r_ax=10;
  reg.r_bx=20;
  intr(0x10,reg);
}
```

Menampilkan Tulisan ke Layar

Dalam menampilkan teks pada layar monitor, digunakan fungsi printf. Adapun teks yang akan dikeluarkan diletakkan dalam tanda ". Dalam C tidak dikenal pembedaan seperti writeln dan write.

Listing Program dalam Turbo Pascal

```
Uses Crt,Dos;

Begin
  Write('halo');
End.
```

Listing Program dalam C

```
#include <stdio.h>

void main()
{
  printf("halo");
}
```

Dalam C, fungsi printf mengenal bermacam-macam karakter format yang berguna untuk mengatur tampilan teks pada layar. Karakter format yang paling penting ialah "/n". Karakter format ini digunakan untuk memindahkan kursor pada baris berikutnya.

Listing Program dalam Turbo Pascal

```
Writeln('  halo  ');
Writeln('  world  ');
```

Listing Program dalam C

```
printf("  halo  \n  world  \n");
```

Begitu menerima karakter format \n, C akan memindahkan kursor pada baris berikutnya, sehingga teks berikutnya yang akan ditampilkan di layar akan diletakkan pada baris berikutnya.

Seperti halnya pada Turbo Pascal, C juga mengenal fungsi GOTOXY yang berguna untuk meletakkan teks pada koordinat tertentu di layar. Adapun fungsi ini dimiliki oleh library conio.h. Jadi jangan lupa untuk mendefinisikan library ini dengan #define.

Listing Program dalam Turbo Pascal

```
gotoxy(10,5);write('halo');
```

Listing Program dalam C

```
gotoxy(10,5);printf("halo");
```

Cara menampilkan harga dari variabel ke layar dalam C jauh berbeda dengan Turbo Pascal. Fungsi yang digunakan tetap printf tetapi dengan parameter berbeda. Untuk menampilkan harga dari sebuah variabel kita harus memberitahukan apa jenis variabel tersebut melalui parameter pertama dari C.

Formatnya ialah sebagai berikut:

printf("type1 type2 type3",var1,var2,var3);

Adapun format-format dari variabel tersebut adalah sebagai berikut.

Tabel 1.4 Format-format Variabel

Variabel format	Arti	Variabel yang biasanya direpresentasikan
%d	signed desimal	Int, char, long, short
%u	unsigned desimal	Int, char, long, short
%c	karakter	Char
%s	string	Array of char (string)
%f	pecahan	Float, double, single
%o	signed octal	Int, char, long, short
%X	unsigned hexa	Int, char, long, short
%p	pointer	[var type] *

Listing Program dalam Turbo Pascal

```
Uses Crt,Dos;

Var a,b:Integer;
    c  :Real;
    d  :String[20];

Begin
  a:=5;b:=3;c:=5.3;
  d:='halo';
  writeln(a,b,c,d);
End.
```

Listing Program dalam C

```
#include <stdio.h>

int   a=5,b=3;
float c=5.3;
char  d[20];

void main()
{
  strcpy(d,"halo");
  printf("%d %d %f %s \n",a,b,c,d);
}
```

Perhatikan, %d pertama menunjukkan variabel a akan ditampilkan berupa angka desimal, %d kedua menunjukkan variabel b, %f menunjukkan variabel c dalam pecahan dan %s menunjukkan variabel d ialah array of char (string).

Pada printf, di tempat didefinisikannya jenis variabel dapat juga disisipkan teks yang akan ditampilkan pada monitor.

Listing Program dalam Turbo Pascal

```
writeln('Var A = ',a,'Var B = ',b);
```

Listing Program dalam C

```
printf("Var A = %d Var B = %d \n",a,b);
```

Secara tidak langsung variabel format juga menunjukkan tempat di mana variabel tersebut akan ditampilkan.

Untuk menghapus layar digunakan fungsi clrscr yang juga dimiliki oleh library conio.h.

Membaca Input melalui keyboard

Fungsi yang digunakan untuk membaca input melalui keyboard ada beberapa macam. Yang pertama ialah membaca sebuah karakter yang diinputkan melalui keyboard yaitu fungsi getch.

Listing Program dalam Turbo Pascal

```
Uses Crt,Dos;

Var ch:char;

Begin
  repeat
    repeat                  // loop sampai ada tombol yang ditekan.
    until keypressed;
    ch:=readkey;
  until ch=#27;       // keluar dari loop jika tombol ESC ditekan.
End.
```

Listing Program dalam C

```
#include <stdio.h>

char ch;

void main()
{
  do
  {
    while (kbhit()==0)    // loop sampai ada tombol yang ditekan.
    {
    }
    ch=getch();
  } while (ch!=27);        // tetap loop jika bukan tombol ESC yang ditekan.
}
```

Seperti yang telah kita lihat di atas, fungsi kbhit sama persis dengan fungsi keypressed yaitu mengeluarkan harga true atau 1 jika ada tombol keyboard yang ditekan.

Fungsi yang kedua ialah scanf. Fungsi ini digunakan untuk menerima line input (menunggu input dari keyboard yang panjangnya bisa lebih

dari satu karakter, input selesai dengan ditekannya tombol enter). Fungsi ini sama persis dengan fungsi read atau readln dari Turbo Pascal. Tetapi sama seperti printf, pada scanf kita diharapkan memberitahukan format variabel yang akan ditampilkan di layar. Adapun variabel format yang digunakan sama persis dengan apa yang telah kita bahas sebelumnya.

Listing Program dalam Turbo Pascal

```
Uses Crt,Dos;

Var a:Integer;
    b:String[50];

Begin
  write('Masukkan harga A (desimal) = ');
  readln(a);
  write('Masukkan harga B (string)  = ');
  readln(b);
End.
```

Listing Program dalam C

```
#include <stdio.h>

int  a;
char b[50];

void main()
{
  printf("Masukkan harga A (desimal) = ");
  scanf("%d",&a);
  printf("Masukkan harga B (string)  = ");
  scanf("%s",b);
}
```

Yang perlu diingat:

- Segala variabel kecuali variabel string yang digunakan untuk menerima input dari keyboard dalam fungsi scanf harus diawali dengan tanda & yang memiliki arti sama dengan alamat dari variabel tersebut. Karena kita mengharapkan fungsi scanf mengisikan harga ke dalam variabel tersebut maka kita harus mengirimkan variabel tersebut dengan "pass by refference". Hal ini akan kita bahas lebih jelas pada bagian Pembuatan subroutine.

- Pada parameter pertama dari fungsi scanf yaitu yang berisi variabel format, kita tidak dapat menyisipkan teks untuk ditampilkan di layar.
- Kita dapat melakukan pembacaan beberapa variabel sekaligus dalam satu scanf sama seperti pada Turbo Pascal. User harus menginputkan angka-angka tersebut dengan dipisahkan oleh tanda spasi. Misalnya:

  scanf("%d %c %f",&a,&b,&c);

1.7 Penggunaan Subroutine

Pembuatan Procedure

Seperti halnya dalam Turbo Pascal, dalam C subroutine dibedakan menjadi dua macam yaitu procedure dan function.

Procedure dideklarasikan dengan istilah void (tidak menghasilkan harga).

Listing Program dalam Turbo Pascal

```
Procedure Testing;
  Begin
  End;
```

Listing Program dalam C

```
void Testing()
{
}
```

Dalam C, deklarasi variabel dalam subroutine dilakukan pada awal subroutine. Sama persis seperti Turbo Pascal. Juga limit dari berlakunya variabel tersebut ialah hanya di dalam subroutine tersebut.

Listing Program dalam Turbo Pascal

```
Procedure Testing;
Var A:Integer;
    B:Real;
  Begin
  End;
```

Listing Program dalam C

```
void Testing()
{
  int   A;
  float B;
}
```

Adapun cara pemanggilan dari procedure ini ialah sebagai berikut.

Listing Program dalam C

```
void main()
{
  Testing();
}
```

Pembuatan Function

Function adalah subroutine yang menghasilkan harga. Cara mendeklarasikan sebuah function ialah dengan mendeklarasikannya sebagai jenis variabel yang akan dihasilkannya.

Listing Program dalam Turbo Pascal

```
Function IsEmpty(a:integer):boolean;
  Begin
    if a=0 then IsEmpty:=true else IsEmpty:=false;
  End;
```

Listing Program dalam C

```
int IsEmpty(int a)
{
   if (a==0)
     return 1;          // Di baris ini program akan meninggalkan function.
   else
     return 0;          // Di baris ini program akan meninggalkan function.
}
```

Adapun cara pemanggilan function ini ialah:

```
void main()
{
  int index=0;
  if (IsEmpty(index)==1)
```

```
  {
  }
}
```

atau

```
void main()
{
  int answer;
  int index=0;
  answer=IsEmpty(index);
  if (answer==1)
  {
  }
}
```

atau

```
void main()
{
  int index;
  IsEmpty(index);
}
```

Pada contoh ketiga, hasil dari function IsEmpty tidak diterima sama sekali. Ini bukan error, karena C tidak selalu mengharapkan kita mempergunakan hasil dari sebuah function.

Mengirimkan Parameter ke Subroutine

Seperti halnya Turbo Pascal, dalam C dikenal dua macam cara untuk mengirimkan parameter ke subroutine.

Pass by Value

Di sini variabel dikirimkan sebagai kopi dari variabel yang asli. Sehingga jika dilakukan perubahan harga terhadap variabel tersebut, maka yang berubah hanyalah kopi dari variabel itu, sedangkan variabel yang asli tetap tidak berubah.

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var a,b:Integer;
```

```
Procedure Testing(a,b:Integer)
  Begin
    a:=20;
  End;

Begin
  a:=5;
  Testing(a,50);
  Writeln(a);
End.
```

Listing Program dalam C

```
#include <stdio.h>

int a=5,b;

void Testing(int a,int b)
{
  a=20;
}

void main()
{
  Testing(a,50);
  printf("%d",a);
}
```

Yang perlu diingat:

- Dalam mendeklarasikan beberapa variabel sebagai parameter dari subroutine, setiap variabel harus dideklarasikan secara terpisah.

  ```
  Testing(int a,b)        // dianggap sebagai error.
  ```

  Yang benar ialah:

  ```
  Testing(int a, int b);
  ```

Pass by Refference

Di sini variabel dikirimkan berupa alamat memory dari variabel tersebut. Sehingga apabila terjadi perubahan harga dari variabel tersebut di dalam subroutine, variabel asli akan ikut berubah.

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var a,b:Integer;

Procedure Testing(Var a; b:Integer)
  Begin
    a:=20;
  End;

Begin
  a:=5;
  Testing(a,50);
  Writeln(a);
End.
```

Listing Program dalam C

```
#include <stdio.h>

int a=5,b;

void Testing(int * a,int b)
{
  *a=20;
}

void main()
{
  Testing(&a,50);
  printf("%d",a);
}
```

Pada main program dari C, terlihat bahwa yang dikirimkan lewat parameter ialah alamat dari variabel a (&a). Di dalam subroutine Testing, penggunaan variabel a harus diawali dengan tanda "*" karena variabel a di dalam subroutine tersebut sebenarnya ialah alamat global variabel a. *a mewakili isi alamat tersebut. Memang penggunaan pointer di C jauh lebih rumit daripada Turbo Pascal. Penjelasan lebih lanjut akan dibahas pada bagian pointer.

Sekarang bagaimana cara mengirimkan array lewat parameter?

Perhatikan contoh program berikut.

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Type MDat=Array[0..100] of Integer;

Var M:MDat;

Procedure(Buffer:MDat)
  Begin
    Buffer[10]:=3;
  End;

Begin
  Testing(M);
End.
```

Listing Program dalam C

```
#include <stdio.h>

int M[100];

void Testing(int Buffer[100])
{
  Buffer[10]=3;
}

void main()
{
  Testing(M);
}
```

Dalam pengiriman array, tidak diperlukan tanda & karena variabel array sebenarnya adalah variabel pointer. Sehingga dengan mengirimkan variabel tersebut sama dengan mengirimkan alamat array. Juga pada saat menerimanya kita dapat menggunakannya seperti array biasa. Ingat pengiriman array selalu "Pass by Refference" jadi setiap kali kita mengubah data di dalam array tersebut, array asli akan ikut berubah.

Nah, sekarang bagaimana mengirimkan string lewat parameter?

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;
```

```
Var a:String[50];

Procedure Testing(a:String);
  Begin
    Writeln(a);
  End;

Begin
  a:='halo';
  Testing(a);
End.
```

Listing Program dalam C

```
#include <stdio.h>

char a[50];

void Testing(char a[50])
{
  printf("%s",a);
}

void main()
{
  strcpy(a,"halo");
  Testing(a);
}
```

Seperti yang kita lihat di atas, karena string adalah array of char maka string diperlakukan sama dengan array. Pengiriman string juga termasuk pass by refference.

Sebenarnya penerimaan string tersebut dapat dilakukan dengan cara:

```
void Testing (char * a)
{
  printf("%s",a);
}
```

Karena sebenarnya string adalah array of char. Sedangkan variabel array disimpan sebagai pointer dari elemen array tersebut yang dalam situasi ini ialah char.

Memang C jauh lebih kompleks dan lebih bebas daripada Turbo Pascal. Terutama dalam hal pointer. Untuk itu sangat penting dalam memahami penggunaan pointer. Pembahasan tentang pointer lebih mendalam dapat diikuti pada bagian penggunaan pointer.

Prototype dari Subroutine

Dalam C, dikenal istilah prototype. Tujuan dari prototye ini ialah untuk mendefinisikan subroutine-subroutine yang kita pergunakan. Sebenarnya ini bukanlah kewajiban, jika kita telah yakin bahwa setiap subroutine berada di atas subroutine di mana subroutine tersebut dipanggil (sama seperti Turbo Pascal). Dengan prototype, peletakan subroutine bebas. Selain itu kegunaan prototype ialah untuk mendefinisikan subroutine-subroutine library/unit yang kita buat. Hal ini akan dijelaskan lebih lanjut pada bagian pembuatan library/unit.

Pada umumnya prototype diletakkan di antara deklarasi variabel dan bagian implementasi program.

```
#include <stdio.h>

// deklarasi variabel ----------------------------------
int a,b;

// definisi prototype ----------------------------------
void DrawCircle(int x,int y,int r,char color);
void DrawLine(int x1,int y1,int x2,int y2,char color);

// implementasi program --------------------------------

void DrawCircle(int x,int y,int r,char color)
{
}

void DrawLine(int x1,int y1,int x2,int y2,char color)
{
}

void main()
{
}
```

Prototype subroutine dideklarasikan persis seperti bagaimana subroutine tersebut dibuat pada bagian implementasi dari program. Sebenarnya nama variabel tidak perlu diikutsertakan dalam definisi prototype, tetapi pemberian nama variabel lebih dianjurkan karena mempermudah kita mengetahui apa kegunaan dari masing-masing parameter.

Prototype tanpa nama variabel dapat dituliskan sebagai berikut:

```
void DrawCircle(int, int, int, char);
```

1.8 File

Text File

Seperti halnya Turbo Pascal, dalam C dikenal dua macam file yaitu binary file dan text file. Dalam Turbo Pascal kedua jenis file ini direpresentasikan dalam jenis variabel yang berbeda (FILE untuk binary file dan TEXT untuk text file). Dalam C kedua jenis file ini direpresentasikan dalam jenis variabel yang sama yaitu FILE.

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var Dat:Text;
    a  :integer;

Begin
  a:=52;
  assign(Dat,'data.dat');
  rewrite(Dat);
  write(Dat,a);
  writeln(Dat,' text');
  close(Dat);
End.
```

Listing Program dalam C

```
#include <stdio.h>

FILE * Dat;
int    a=52;

void main()
{
  Dat=fopen("data.dat","w");
  fprintf(Dat,"%d",a);
  fprintf(Dat," text \n");
  fclose(Dat);
}
```

Untuk membuka file, kita harus menyebutkan jenis operasi yang akan kita lakukan terhadap file tersebut. Hal ini sama persis seperti pada Turbo Pascal. Hanya pada Turbo Pascal, digunakan fungsi Rewrite dan Reset untuk menentukan jenis operasi tersebut setelah file

ditunjuk oleh fungsi assign. Dalam C, pada saat kita membuka file tersebut, sebagai parameter kedua, kita sebutkan jenis operasi yang akan dilakukan yaitu "w" berarti write dan "r" berarti read.

Untuk menuliskan data ke dalam file jenis TEXT digunakan fungsi fprintf. Penggunaan fungsi fprintf ini sama persis dengan penggunaan fungsi printf. Satu-satunya perbedaan ialah disebutkannya nama variabel file sebagai parameter pertama. Pada contoh di atas, variabel file yang kita gunakan ialah Dat.

Nah, sekarang kita lihat bagaimana cara membaca data dari TEXT file.

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var Dat:Text;
    a  :integer;
    txt:string[20];

Begin
  assign(Dat,'data.dat');
  reset(Dat);
  read(Dat,a);
  readln(Dat,txt);
  close(Dat);
End.
```

Listing Program dalam C

```
#include <stdio.h>

FILE * Dat;

int a;
char txt[20];

void main()
{
  Dat=fopen("data.dat","r");
  fscanf(Dat,"%d",&a);
  fscanf(Dat,"%s",txt);
  fclose(Dat);
}
```

Fungsi fscanf digunakan untuk membaca data dari file. Adapun penggunaannya sama persis dengan scanf, hanya saja kita perlu

memberitahukan variabel file yang akan dibuka lewat parameter pertama dari fscanf.

Binary File

Seperti yang telah kita bahas, jenis variabel file yang kita gunakan untuk BINARY dan TEXT file dalam C adalah sama. Bahkan untuk membuka dua jenis file ini digunakan fungsi yang sama yaitu fopen. Adapun perbedaannya ialah ditambahkannya karakter "b" yang berarti binary dalam parameter kedua dari fopen. Sehingga untuk membuka file data.dat untuk write sebagai binary file adalah seperti berikut:

```
dat=fopen("data.dat","rb");
```

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var Dat:File;
    a  :integer;

Begin
  assign(Dat,'data.dat');
  reset(Dat,1);
  blockread(Dat,a,2);
  close(Dat);
End.
```

Listing Program dalam C

```
#include <stdio.h>

FILE * Dat;

int a;

void main()
{
  Dat=fopen("data.dat","rb");
  fread(&a,2,1,Dat);
  fclose(Dat);
}
```

Untuk membaca data dari binary file digunakan fungsi fread. Adapun format dari fungsi ini adalah sebagai berikut:

```
fread(buffer, panjang, unit, variabel file);
```

buffer : alamat memory di mana data akan disimpan.
Panjang : ialah panjang data dalam satuan unit yang akan dibaca.
unit : adalah unit yang dipergunakan oleh parameter panjang.
variabel file : adalah variabel file dari file yang akan dibaca.

Pada contoh di atas, parameter panjang = 2 karena integer berukuran 2 byte. Sedangkan unit = 1.

Untuk menulis data ke binary file digunakan fungsi fwrite. Adapun format dari fungsi ini ialah:

```
fread(buffer, panjang, unit, variabel file);
```

buffer : alamat memory di mana data yang akan ditulis disimpan.
Panjang : ialah panjang data dalam satuan unit yang akan disimpan.
Unit : adalah unit yang dipergunakan oleh parameter panjang.
variabel file : adalah variabel file dari file di mana data akan disimpan.

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var Dat:File;
    a  :integer;

Begin
  a:=52;
  assign(Dat,'data.dat');
  rewrite(Dat,1);
  blockwrite(Dat,a,2);
  close(Dat);
End.
```

Listing Program dalam C

```
#include <stdio.h>
```

```
FILE * Dat;

int a=52;

void main()
{
  Dat=fopen("data.dat","wb");
  fwrite(&a,2,1,Dat);
  fclose(Dat);
}
```

1.9 Pembuatan Unit

Cara memanggil unit

Sama halnya seperti Turbo Pascal, dalam C dikenal istilah unit. Unit ialah subroutine-subroutine yang dikumpulkan dalam satu file. Berikutnya jika kita hendak mempergunakan salah satu subroutine dari library tersebut, kita tinggal mendefinisikan library tersebut pada awal program. (Dalam C digunakan fungsi #include <nama library> sedangkan dalam Turbo Pascal digunakan fungsi Uses nama library.)

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos, MyUnit;

Begin
End.
```

Listing Program dalam C

```
#include <stdio.h>
#include "MyUnit.h"

void main()
{
}
```

Dalam C, unit yang telah disediakan oleh C didefinisikan dengan menggunakan #include <....> sedangkan unit yang kita buat sendiri didefinisikan dengan menggunakan #include "....". Sedangkan nama unit yang harus kita isikan di antara tanda " ialah nama prototype file dari unit yang kita buat lengkap dengan directory tempat file tersebut

berada. Jika directory tempat unit tersebut berada sama dengan directory tempat main program berada, kita tidak perlu menyebutkan nama directory.

Pembuatan Unit dalam C

Dalam C, pembuatan unit meliputi pembuatan dua buah file yaitu prototype file dan implementation file. Prototype file berisi prototype seluruh subroutine yang disediakan oleh unit tersebut, yang dapat dipergunakan oleh program yang mendefinisikan unit tersebut dan deklarasi variabel-variabel dan konstanta-konstanta yang akan dipergunakan oleh unit tersebut. File ini memiliki nama sama dengan nama unit, hanya saja berekstensi .h. Sedangkan implementation file berisi subroutine-subroutine yang disediakan oleh unit tersebut dan berekstensi .c.

Listing Program dalam Turbo Pascal

Unit file: VGACAD.PAS

```
Unit VGACAD;

INTERFACE

Var X,Y,Z:Integer;

Procedure PutPixel(X,Y:Integer;Col:Byte);
Function GetPixel(X,Y:Integer):Boolean;

IMPLEMENTATION

Procedure PutPixel(X,Y:Integer;Col:Byte)
  Begin
    Mem[$A000:Y*320+X]:=Col;
  End;

Function GetPixel(X,Y:Integer):Boolean;
  Begin
    GetPixel:=Mem[$A000:Y*320+X];
  End;
```

Listing Program dalam C

Prototype file: vgacad.h

```
#define SCREEN (char *)(0x0A0000)
```

```
int X,Y,Z;

void PutPixel(int X,int Y,char Col);
int  GetPixel(int X,int Y);
```

Implementation file: vgacad.c

```
#include <stdio.h>
#include <mem.h>
#include "vgacad.h"      // definisikan prototype dari unit.

void PutPixel(int X,int Y,char Col)
{
  *(SCREEN+Y*320+X)=Col;
}

int GetPixel(int X,int Y)
{
  return *(SCREEN+Y*320+X);
}
```

1.10 Penggunaan Pointer

Penggunaan pointer dalam C jauh lebih bebas dan lebih jelas dibandingkan dengan Turbo Pascal. Variabel pointer berisi alamat memory tempat suatu data disimpan. Pada C, variabel pointer dideklarasikan dengan menyebutkan jenis data yang disimpan dalam alamat memory tersebut. Misalnya pada alamat memory tersebut akan disimpan data integer, maka variabel pointer p akan dideklarasikan sebagai:

```
int * p;
```

Sedangkan untuk mengisikan data ke dalam alamat memory yang ditunjuk oleh variabel p (misalnya dengan memasukkan angka 5) ialah dengan cara:

```
*p = 5;
```

Demikian juga untuk segala operasi yang membutuhkan akses dari data yang ditunjuk oleh variabel p, selalu diwakili oleh *p. Misalnya untuk menampilkan isi data ke layar:

```
printf("%d",*p);
```

Seperti halnya dalam Turbo Pascal, dalam C, variabel pointer juga dapat digunakan untuk menyimpan alamat byte pertama dari sejumlah data pada alamat memory. Misalnya kita memiliki 200 byte data yang disimpan dalam alamat memory 10000 hingga 10199, kita cukup mendeklarasikan sebuah variabel pointer bertipe char (misalnya variabel p) yang menunjuk pada alamat memory 10000.

Mengisi Alamat Memory dengan Data Lewat Pointer

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var A:Pointer;

Begin
  GetMem(A,100);          { Pesan memory 100 byte dengan awal
                  lokasi memory disimpan dalam variabel A. }
  FillChar(A^,100,0);              { Isi alamat memory mulai A
                                       Hingga A+99 dengan 0. }
End.
```

Listing Program dalam C

```
#include <stdio.h>
#include <mem.h>

char * a;

void main()
{
  a=(char *)malloc(100)              // Pesan memory 100 byte
                                  // dengan awal lokasi memory
                                 // disimpan dalam variabel a.
                     // Char menunjukkan jenis data yang akan
                           // disimpan dalam alamat memory a.
  memset(a,0,100);                 // Isi alamat memory mulai a
                                       // hingga a+99 dengan 0.
}
```

Segala fungsi yang berkenaan dengan memory disediakan oleh library mem.h.

Menduplikasi Data dalam Alamat Memory ke Alamat Memory yang Lain Lewat Pointer

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;

Var A,B:Pointer;

Begin
  GetMem(A,100);        { Pesan memory 100 byte dengan awal
                  lokasi memory disimpan dalam variabel A. }
  GetMem(B,100);        { Pesan memory 100 byte dengan awal
                  lokasi memory disimpan dalam variabel B. }
  Move(B^,A^,100);          { Duplikat data dari alamat A ke
                              alamat B sejumlah 100 byte. }
End.
```

Listing Program dalam C

```
#include <stdio.h>
#include <mem.h>

char * a;
char * b;

void main()
{
  a=(char *)malloc(100)              // Pesan memory 100 byte
                                  // dengan awal lokasi memory
                                  // disimpan dalam variabel a.
                    // Char menunjukkan jenis data yang akan
                 // yang akan disimpan dalam alamat memory a.
  b=(char *)malloc(100);
  memcpy(b,a,100);             // Duplikasi data dari alamat a
                                // alamat b sejumlah 100 byte.
}
```

Menduplikasi Data dari Variabel ke Alamat Memory Lewat Pointer

Listing Program dalam Turbo Pascal

```
Uses Crt, Dos;
```

```
Var A   :Pointer;
    Num:Integer;

Begin
  GetMem(A,100);
  Num:=5;
  Move(A^,Num,2);          // Duplikasi data dari variabel Num
                              // ke alamat A sejumlah 2 byte
                                 // karena integer = 2 byte.
End.
```

Listing Program dalam C

```
#include <stdio.h>
#include <mem.h>

int     Num;
char * A;

void main()
{
  A=(char *)malloc(100);
  Num=5;
  memcpy(A,&Num,2);       // Duplikasi data dari variabel Num
                             // ke alamat A sejumlah 2 byte.
}
```

Di sini terlihat perbedaan antara Turbo Pascal dan C. Dalam Turbo Pascal, kita cukup menyebutkan variabel yang akan diduplikat, sedangkan dalam C, kita harus menyebutkan alamat variabel yang akan kita duplikat (digunakannya tanda &). Kenapa pada contoh listing program sebelumnya kita tidak perlu menyebutkan alamat variabel b dan a pada saat kita melakukan memcpy? Karena kedua variabel ini adalah variabel pointer. Variabel ini sudah mencerminkan alamat dari data sehingga kita tidak perlu menggunakan tanda &. Sedangkan variabel Num pada contoh di atas mencerminkan data (bukan mencerminkan alamat) untuk itu diperlukannya tanda &.

Kelebihan Pointer di C

Salah satu kelebihan sistem pointer di C ialah kebebasan dalam menggunakan variabel pointer tersebut. Misalnya kita memiliki data pada lokasi memory 10000 sejumlah 100 byte. Variabel yang menunjuk pada awal lokasi memory tersebut ialah variabel p. Jika

kita hendak mengakses data di lokasi 10001 kita dapat melakukan langkah berikut:

```
*(p+1)=5;
```

Dalam Turbo Pascal, hal ini tidak mungkin untuk dilakukan.

BAB 2

PEMROGRAMAN DENGAN WATCOM C

2.1 Pengenalan Watcom C dan Protected Mode

Watcom C sama halnya seperti C compiller yang lain, seperti Borland C, Turbo C. Tetapi Watcom C ini memiliki keunggulan yang jauh lebih tinggi dibandingkan dengan compiller C yang lain dalam kelas DOS.

Salah satu keunggulan compiller ini ialah digunakannya protected mode dalam proses compile-nya. Dengan ini maka untuk setiap alokasi memory yang kita lakukan dapat melebihi 64 kbytes. Juga batasan ukuran program beserta variabelnya dapat melebihi batas 64 kbytes.

Apa itu protected mode? Apa itu batasan 64 kbytes? Ok... ok... akan saya jelaskan berikut ini.

Pemrograman pada komputer dapat dibedakan menjadi dua macam yaitu Protected mode dan Real Mode. Programming yang biasa kita lakukan dengan Turbo Pascal, Borland C, Turbo C, semuanya menggunakan Real Mode.

Pada Real Mode, RAM komputer dibagi menjadi segment-segment di mana masing-masing segmentnya berukuran 64 kbytes. Kenapa? Programming real mode diciptakan pada Zaman komputer masih bertipe XT, AT dan 286. Karena ukuran variabel yang dikenal oleh komputer bertipe tersebut maksimum ialah variabel word yang memiliki radius dari 0 hingga 65536 = 64 kbytes. RAM pada komputer standar pada saat itu ialah 640 kbytes yaitu sepuluh kali lipat lebih besar dari radius variabel word. Sedangkan untuk menyimpan suatu alamat memory (variabel pointer) dibutuhkan variabel yang memiliki radius sama dengan ukuran memory yang dipunyai. Tentu saja hal ini tidak mungkin karena variabel terbesar hanya memiliki radius sepuluh kali lipat lebih kecil dari ukuran memory yang dimiliki.

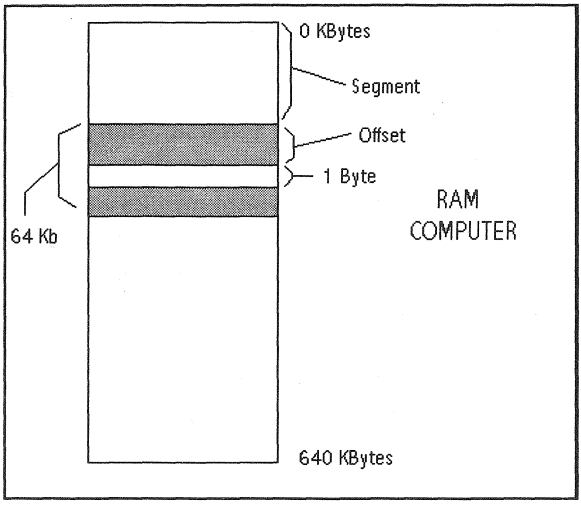
Bila Anda masih bingung, coba bayangkan jika suatu jalan memiliki nomor antara 1 hingga 200. Sedangkan pada amplop surat yang digunakan untuk mengirim hanya disediakan tempat dua digit/angka untuk nomor rumah. Nomor rumah maksimum yang dapat dituliskan ialah 99. Kita tidak dapat menuliskan nomor rumah 100 ke atas. Masalah ini sama halnya dengan penjelasan di atas.

Sebagai pemecahan masalah ini maka digunakannnya sistem segment-offset yaitu pembatasan pengaksesan memory sebesar 64 kbytes. Sehingga sebelum kita hendak mengakses sebuah alamat memory di RAM, kita harus menentukan sebuah segment yang berukuran 64 kbytes di mana alamat memory yang hendak kita akses berada di dalam segment tersebut. Jadi untuk mengakses sebuah alamat memory, digunakan dua variabel word. Variabel yang pertama yaitu untuk menunjukkan segment mana yang hendak dituju, sedangkan variabel word yang kedua menunjukkan offset/index dari segment tersebut.

Seperti yang dilukiskan pada Gambar 2.1, segment menunjukkan alamat start dari segment yang hendak kita akses. Sedangkan Offset menunjukkan index di mana lokasi memory yang hendak kita akses berada relatif terhadap segment. Jika lokasi memory yang hendak kita akses tepat berada pada awal segment, maka offset/index akan berharga 0. Dengan ini variabel offset sebagai index hanya perlu memiliki radius antara 0 hingga 64 kbytes yang memungkinkan untuk disimpan dalam variabel word.

Nah, sekarang masalahnya bagaimana dengan variabel segment. Variabel segment bisa saja diperlukan untuk menunjuk alamat 600 kbytes yang tentu saja tidak cukup untuk disimpan dalam variabel word. Untuk ini variabel segment memiliki skala 16 kali lebih besar dari variabel offset. Alamat memory yang sebenarnya ditunjuk oleh

variabel segment ialah 16 x dari harga yang disimpan dalam variabel tersebut. Misalnya jika variabel segment berharga 100 maka Alamat yang ditunjuk sebenarnya ialah 100 x 16 = 1600. Dengan ini, radius yang tadinya berkisar antara 0 hingga 64 kbytes menjadi 0 hingga 640 kbytes. Dengan teknik ini, segment hanya dapat dibentuk dari kelipatan angka 16 yaitu 0, 16, 32, 48, 64 dan seterusnya.



Gambar 2.1 Segment dan Offset pada RAM

Nah, karena pada satu saat, kita hanya dapat mengakses 64 kbytes memory, maka setiap mengadakan alokasi memory kita hanya diberi maksimal 64 kbytes. Juga karena program kita beserta variabel akan diletakkan dalam segment, maka maksimal berukuran 64 kbytes.

Memang akal Bill Gates memang canggih dengan menggunakan teknik ini dalam DOS, tetapi semenjak dikeluarkannya 386, 486, pentium dan lain-lainnya di mana kebutuhan memory semakin meningkat, teknik segment-offset ini membawa malapetaka bagi para programmer. Dengan dikeluarkannya tipe komputer baru dengan variabel yang memiliki skala lebih besar dari variabel word, maka sekarang tidak diperlukan lagi sistem segment dan offset ini karena variabel long integer (32 bit) memiliki radius antara 0 hingga 2 Giga bytes. Sehingga DOS versi baru mendukung digunakannya variabel pointer 32 bit tetapi penggunaannya masih cukup rumit. Inilah yang disebut dengan programming di protected mode.

Untuk selanjutnya pemrograman dengan teknik segment offset akan disebut sebagai pemrograman real mode, dan pemrograman dengan

32 bit pointer akan disebut sebagai pemrograman protected mode.

Karena sistem programming real mode sangat tidak efisien dibandingkan protected mode, maka programmer cenderung meninggalkan real mode. Tetapi masih banyak programmer yang tetap menggunakan real mode karena pemrograman di protected mode masih sangat rumit.

Semenjak saat itu pemrograman di komputer hingga sekarang masih dibedakan menjadi dua macam, karena tentu saja operating-operating system yang baru harus tetap menyediakan sistem real mode, walaupun hal ini tidak efisien. Karena jika tidak, program–program terdahulu yang diciptakan sebelum adanya protected mode tidak akan jalan.

Masih belum tidur kan? Bagus.... Sekarang kita kembali ke Watcom kita yang tercinta. Dalam masa-masa kesulitan, para programmer memprogram dalam protected mode, Tenberry software menciptakan compiller Watcom C plus DOS extender (DOS tambahan) yang diberi nama DOS4GW (DOS for Game Wizard) untuk memperingan pekerjaan para programmer game. Kenapa kok ditujukan kepada programmer game? Karena programmer game terkenal pelit dalam masalah optimasi kecepatan, sehingga seluruh program harus dibuat seefisien mungkin, yang berarti harus banyak berkutat dalam protected mode.

Watcom C beserta DOS4GW memberikan dukungan 32 bit protected mode secara otomatis, tanpa perlu campur tangan programmer sehingga secara otomatis apa yang kita program akan diimplementasikan dalam protected mode.

Salah satu cara membuktikan bahwa kita telah menggunakan protected mode dalam Watcom C, ialah dengan mencoba cuplikan program di bawah ini:

```
char data[1000000];
```

Pada Turbo C atau Borland C, tentu saja hal ini akan mengakibatkan error karena variabel menggunakan memory 1 Mega Bytes, yaitu lebih dari satu segment (64 kbytes). Jangan kaget.... Pada Watcom C, jika Anda memiliki memory 2 Giga Bytes, cuplikan program di bawah ini masih tetap jalan:

```
char data[1000000000];
```

Atau alokasi dynamic seperti di bawah ini akan jalan di Watcom C tapi tidak akan jalan di compiller C real mode.

```
Char * data;
Data = (char *)malloc(500000);
```

Karena hal inilah, para programmer game mulai laris menggunakan Watcom C. Misalnya game DOOM, Duke Nukem 3D, Command and Conquer, WarCraft, Raptor dan masih banyak yang lainnya.

Nah, sekarang kita telah mengerti keunggulan-keunggulan Watcom C dibanding compiller C yang lain, apalagi yang kita tunggu? Yuk, kita lihat lebih lanjut penggunaannya.

2.2 Instalasi Watcom C

Watcom C terdiri dari program-program utility dan C library yang berukuran sekitar 250 Mega Bytes. Tenang.... Tenang.... Dari 250 Mega Bytes tersebut, yang biasanya digunakan untuk memprogram dan men-debug program hanyalah 20 Mega Bytes, itupun sudah termasuk help dan contoh-contoh program. Fasilitas yang harus digunakan sebenarnya berukuran 10 Mega Bytes, yaitu ulility dan libraynya.

Adapun Watcom dapat diinstal dalam dua versi yaitu DOS atau Windows. Pada pembahasan kita selanjutnya akan kita gunakan versi DOS, karena ukuran instalasinya jauh lebih kecil.

Pada saat instalasi, jalankan file INSTALL.EXE (jangan memakai file SETUP.EXE karena file ini untuk versi windows). Kemudian jawab pertanyaan-pertanyaan yang ditampilkan hingga Anda mendapati layar dengan pertanyaan, Select Host Operating dan Select Target Operating System.

Host Operating System ialah operating system yang Anda gunakan di mana Anda akan memprogram, sedangkan Target Operating System ialah operating system di mana Anda ingin program hasil compile akan dijalankan.

Pilih DOS pada kedua pertanyaan ini dan pilihlah pernyataan Include 16 bit development. Lanjutkan ke window selanjutnya dengan menekan enter. Pada window selanjutnya akan ditanyakan bagian-bagian apa saja yang akan diinstal. Pilihlah example source code dan Help files on Hard Disk. (Help file ini berukuran sekitar 7 Mega Bytes dan sangat membantu sekali selama pemrograman berlangsung.)

Pada saat akhir akan ditanyakan apakah Anda ingin Watcom C melakukan perubahan pada CONFIG.SYS dan AUTOEXEC.BAT di sistem Anda. Jawab YES karena perubahan yang dilakukan sangat

penting dalam mengatur path environment dari sistem Anda. Tanpa adanya path ini, maka utility-utility dari Watcom tidak akan mengetahui letak C library dalam hard disk.

Nah, selesailah proses instalasi. Sekarang reset komputer Anda untuk memanggil AUTOEXEC dan CONFIG yang baru saja diganti.

2.3 Penggunaan Utility Watcom C

Walaupun Watcom C memiliki banyak utility, hanya beberapa saja yang akan kita gunakan selama pembahasan buku ini. Yaitu WPP386.EXE, WLINK.EXE, WD.EXE dan WHELP.EXE.

Jika AUTOEXEC dan CONFIG sistem kita telah diganti secara otomatis oleh watcom pada saat instalasi, keempat file ini dapat dipanggil pada directory manapun, jadi kita tidak perlu mengikutsertakan directory tempat keempat file ini berada. Hal ini sangat memudahkan proses pemrograman.

WPP386.EXE Compiler

Utility ini digunakan untuk meng-compile listing program. Lain halnya dengan compiler C yang lain ataupun Turbo Pascal, Watcom C memiliki file khusus yang digunakan untuk meng-compile program yang terpisah dari editor di mana listing program kita ketik.

Untuk mengetik dapat digunakan program VI.EXE yang disediakan oleh watcom, atau Anda juga dapat menggunakan text editor yang lain, misalnya EDIT.EXE ataupun editor dari Borland C dan Turbo C. Setelah listing program selesai diketik, kita harus keluar dari text editor tersebut dan meng-compile file listing program dengan WPP386.EXE menjadi object file (berekstensi OBJ). Langkah selanjutnya ialah menciptakan executable file dari object file tersebut dengan menggunakan WLINK.EXE.

Adapun proses-proses di atas dapat dituliskan sebagai berikut:

```
WPP386 /d3 /5s namafile
WLINK system dos4g debug all file namafile name namafile
```

Namafile diisikan dengan nama file yang akan di-compile. Ingat pada saat pemanggilan WPP386 namafile diisikan lengkap dengan ekstensinya, sedangkan pada pemanggilan WLINK, namafile cukup diisi dengan nama file tanpa ekstensi.

Parameter /d3 meminta compiller untuk mengikutsertakan informasi dalam men-debug program yang sudah jadi. Proses men-debug program akan dijelaskan lebih lanjut pada bagian WD.EXE.

Parameter /5s memberitahukan compiller bahwa model stack program ditujukan untuk pentium. Parameter ini harus selalu diikutsertakan dan harus sesuai dengan jenis komputer yang Anda gunakan untuk menjalankan program hasil compile. Adapun jenis-jenis model program yang disediakan ialah:

/3s : model stack program untuk 386.

/4s : model stack program untuk 486.

/5s : model stack program untuk pentium.

Adapun parameter-parameter penting yang lain ialah:

/fp5 : optimasi kecepatan untuk penggunaan floating point pentium.

/od : matikan seluruh teknik optimasi.

/ol+ : optimasi kecepatan untuk penggunaan loop dengan melakukan unroll loop.

Unroll loop adalah teknik membuka sebuah loop yang dinilai akan lebih cepat jika digantikan dengan pengulangan instruksi. Misalnya:

```
For (a=0;a<2;a++)
{
  c++;
}
```

Dengan teknik unroll loop, maka akan menjadi:

```
C++;
C++;
```

Karena bentuk kedua ini dinilai lebih cepat dibandingkan dengan bentuk pertama yang harus melakukan pelacakan harga variabel a, penambahan variabel a dan loop.

WLINK.EXE Linker

Program ini digunakan untuk menciptakan executable file dari file berekstensi OBJ. Seperti yang telah dijelaskan di atas, setelah object file diciptakan dengan WPP386.EXE, object file tersebut selanjutnya harus di-compile lagi menjadi executable file dengan WLINK.EXE.

Adapun caranya ialah sebagai berikut:

```
WLINK system dos4g debug all file namafile name namafile
```

System dos4g menunjukkan digunakannya dos4gw sebagai dos extender.

Debug all file meminta pengisian informasi seluruh unit listing program yang dipakai sebagai informasi debug.

Namafile yang pertama ialah menunjukkan object file (tanpa ekstensi) yang akan di-compile, dan namafile kedua ialah nama executable file (tanpa ekstensi) yang akan dibuat.

Walaupun kemampuan Watcom C sangat andal, tetapi jika program yang kita buat sangat besar sehingga memory komputer kita tidak memungkinkan pada saat compile, kita harus mematikan informasi debug, yaitu dengan cara sebagai berikut:

```
WLINK system dos4g namafile name namafile
```

Dengan ini informasi debug tidak akan diikutsertakan.

Sebenarnya WLINK.EXE ini tidak hanya digunakan untuk menciptakan executable file dari satu file object, melainkan bisa juga dari beberapa object file sekaligus. Hal ini digunakan jika kita hendak menggabungkan external assembly file ke dalam program kita.

Catatan

External assembly ialah program assembly yang kita rakit terpisah dari program utama dan di-compile secara terpisah menjadi object file. Pada akhirnya, object file ini bersamaan dengan program utama yang juga telah berupa object file akan digabungkan menjadi executable file.

Adapun cara pemanggilan WLINK.EXE untuk menciptakan executable file dari beberapa object file ialah sebagai berikut.

Misalnya kita memiliki file VGA.OBJ dan MAIN.OBJ dan kita ingin executable file yang diciptakan bernama TEST.EXE maka cara pemanggilannya ialah sebagai berikut:

WLINK system dos4g VGA MAIN name TEST

WD.EXE Debugger

Utility ini ialah debugger. Tujuannya ialah melacak jalannya executable file yang telah dibuat. Utility ini sama kegunaannya dengan fungsi

submenu DEBUG pada Borland C maupun Turbo C yang memberikan fasilitas Trace, BreakPoint dan lain sebagainya.

Untuk men-debug sebuah program, kita harus memanggil program WD.EXE ini beserta beberapa parameter penting sebagai berikut:

WD /TRAP=RSI namafile

Parameter /TRAP=RSI memberitahukan debugger bahwa file yang akan di-debug menggunakan DOS4GW sedangkan namafile ialah executable file yang akan di-debug lengkap dengan ekstensi (EXE).

Catatan

File yang akan kita debug harus di-compile dengan parameter /d2 atau /d3 dengan menggunakan WPP386.EXE karena debugger WD.EXE membutuhkan informasi debug.

Penggunaan WD.EXE secara lengkap akan dibahas habis-habisan pada bagian akhir bab ini.

WHELP.EXE Help Utility

Utility ini berisi keterangan mengenai fungsi-fungsi C, cara pemrograman C, cara menggunakan Watcom C dan lain sebagainya. Untuk itu kita harus mengikutsertakan parameter untuk memilih topik yang hendak kita cari. Topik-topik itu adalah sebagai berikut:

CGUIDE	:	Menjelaskan instalasi dan dasar dari Watcom C.
CLIB	:	Menjelaskan fungsi-fungsi C.
CPPLIB	:	Menjelaskan fungsi-fungsi C ++.
LGUIDE	:	Menjelaskan penggunaan WLINK.EXE.
PGUIDE	:	Menjelaskan pemrograman di Watcom C.
RESCOMP	:	Menjelaskan penggunaan Watcom Compiler.
TOOLS	:	Menjelaskan fasilitas-fasilitas yang disediakan oleh Watcom.
WD	:	Menjelaskan penggunaan Watcom Debugger.
WPROF	:	Menjelaskan penggunaan Watcom Profiler (mengukur kecepatan).

Adapun cara pemanggilan WHELP.EXE ini adalah sebagai berikut:

WHELP CLIB

Jika kita ingin menampilkan topik tentang fungsi-fungsi C.

Catatan

Topik yang paling sering digunakan biasanya ialah CLIB dan CPPLIB karena berisi fungsi-fungsi yang disediakan oleh Watcom C.

Program Pertama Kita di Watcom C

Nah, agar kita tidak hanya belajar dari membaca saja, sekarang kita akan mempraktekkan membuat sebuah program sederhana dengan utility Watcom C.

Pertama-tama masuklah ke directory TUT1 yang telah diinstal dari disket ke hard disk dan coba cari file bernama TEST.CPP.

Listing Program TEST.CPP

```
/* ------------------------------------------------------------
Nama File   : TEST.CPP
Definisi    : Program mencoba Watcom C
Programmer  : Agustinus Nalwan
------------------------------------------------------------ */
#include <stdio.h>

struct _murid
{
  char nama[50];
  int  nilai;
}

int a;
_murid murid[50];

void display()
{
  int ctr;
  printf("\n");
  for (ctr=0;ctr<50;ctr++)
  {
    printf("nilai = %d\n",murid[ctr].nilai);
  }
}

int main()
{
  for (a=0;a<50;a++)
  {
    murid[a].nilai=a;
  }
```

```
  display();
  return 1;
}
```

Program ini adalah program sederhana yang akan mengisikan nilai 0 hingga 49 ke dalam array murid di variabel nilai. Kemudian akan menampilkan hasilnya pada layar monitor.

Coba compile program ini dengan menggunakan WPP386.EXE seperti yang telah dijelaskan di atas. Pada pentium, pemanggilannya adalah sebagai berikut:

```
WPP386 /d3 /5s TEST.CPP
```

Jika langkah yang Anda lakukan benar, akan terbentuk file bernama TEST.OBJ.

Nah, sekarang coba ciptakan executable file bernama TEST.EXE dengan menggunakan WLINK.EXE seperti yang telah dijelaskan di atas.

```
WLINK system dos4g TEST name TEST
```

Jika langkah yang Anda lakukan benar, maka akan terbentuk file bernama TEST.EXE. Nah, file ini telah siap untuk dijalankan. Selamat atas keberhasilan meng-compile program pertama. (Tidak perlu malu-malu.....) Saat pertama kali berhasil melakukan proses compile (pada saat itu sulit sekali mencari informasi penggunaan utility watcom C), setelah seminggu mencoba, saya berteriak-teriak di dalam kamar apartemen sampai teman satu apartemen saya bangun.

Sekarang coba jalankan program tersebut. Pada layar akan ditampilkan angka mulai 0 hingga 49.

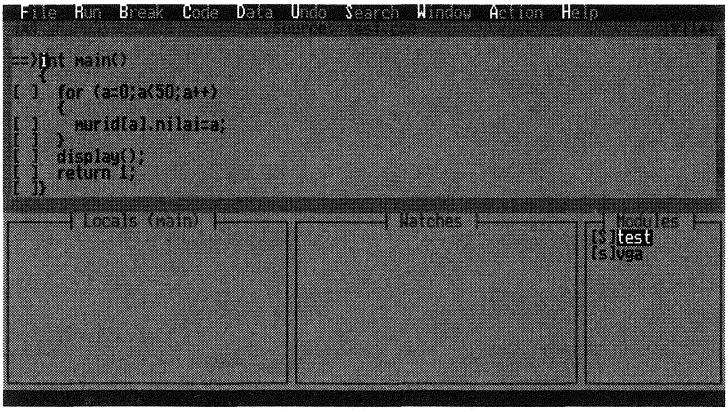
Nah, sekarang kita akan mencoba men-debug program tersebut.

Jalankan Watcom Debugger (WD.EXE) dengan parameter file TEST.EXE. Caranya ialah sebagai berikut:

```
WD /TRAP=RSI TEST.EXE
```

Akan ditampilkan layar seperti pada Gambar 2.2.

Nah, Anda boleh percaya atau tidak, Watcom Debugger ini adalah debugger yang paling canggih yang pernah saya temui. Buktinya ikutilah pembahasan selanjutnya.



Gambar 2.2 Tampilan Awal WD.EXE

Pada saat pertama kali memasuki Watcom Debugger, secara langsung akan ditampilkan listing program dari TEST.EXE. Dengan tanda panah yang menunjuk ke baris:

Int main()

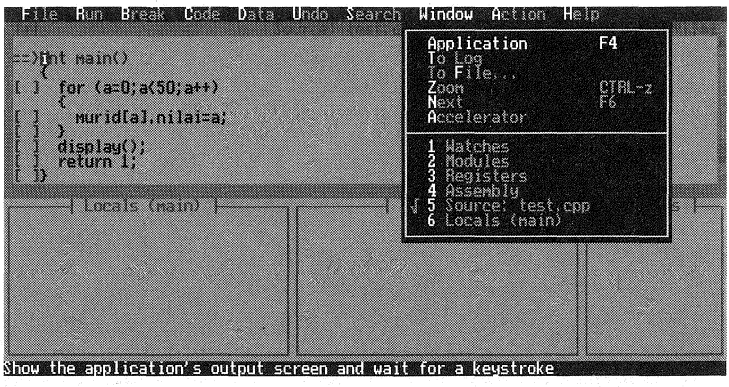
Tanda panah ini menunjukkan baris program yang selanjutnya akan dieksekusi.

Pengaturan Window

Berikutnya akan kita lihat bagaimana cara pengaturan window di layar. Window yang difokus yaitu yang paling atas menampilkan listing program kita. Sedangkan window yang di bawah dengan judul Locals (main) menampilkan variabel lokal yang belum ada sebelum program dijalankan. Nah, sekarang tekan Alt-W, yang akan membuka submenu seperti yang ditampilkan pada Gambar 2.3. Di submenu ini pengaturan window dapat dilakukan.

Untuk memfokus window Locals (main), Anda tinggal menekan angka 6 (seperti yang dituliskan di sebelah kiri submenu Window). Sedangkan untuk memfokus window Source, Anda dapat menekan angka 5.

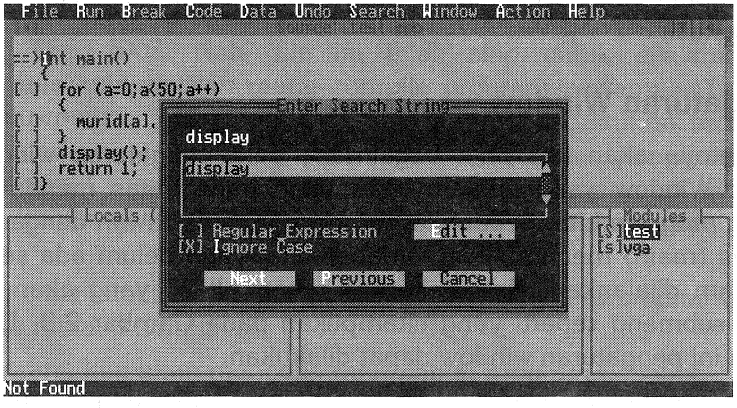
Secara cepat, tanpa perlu membuka submenu Window, kita cukup menekan Alt-1, Alt-2, Alt-3 untuk memilih window yang kita inginkan. Atau juga dapat dilakukan dengan menekan tombol kiri mouse di atas window yang bersangkutan.



Gambar 2.3 Pengaturan Window

Pencarian String

Untuk mencari sebuah string di dalam sebuah window. Misalnya pada window source, kita ingin mencari kata display di dalam listing program, caranya ialah dengan menekan tombol /. Seperti yang ditampilkan pada Gambar 2.4.



Gambar 2.4 Pencarian String

Kemudian isikan kata yang hendak dicari. Pilihan Ignore Case ditujukan untuk mengabaikan huruf besar dan huruf kecil. Kemudian tekan Enter atau pilih tombol NEXT untuk mencari kata display di bawah posisi kursor. Untuk menlakukan pencarian kata display sekali lagi, Anda cukup menekan tombol / dan menekan ENTER. Ingat, pencarian ini tidak akan berakhir, karena setelah angka display terakhir ditemukan, jika kita melakukan pencarian lagi, pencarian akan dilakukan ulang dari awal sehingga angka display yang pertama akan

ditampilkan. Demikian seterusnya.

Fungsi ini sangat penting, apalagi jika program kita sudah sangat panjang dan kita ingin mencari sebuah variabel yang berada di tengah-tengah program.

Pencarian string ini dapat dilakukan pada setiap window yang sedang aktif, jadi tidak hanya berlaku di window source saja. Berikutnya akan dibahas penggunaan pencarian string ini pada window lain.

Pelacakan Jalannya Program

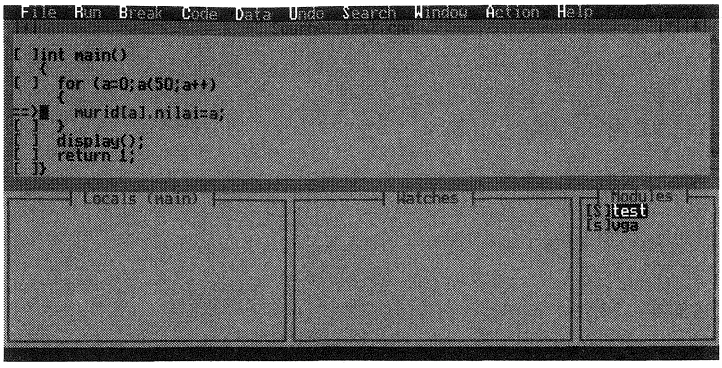
Kita telah membahas beberapa fungsi dari WD.EXE ini. Tetapi apalah arti dari debugger jika tidak dapat menelusuri jalannya program. Tenang.... WD.EXE adalah debugger yang benar-benar lengkap. Untuk menelusuri jalannya program, kita dapat menggunakan fasilitas-fasilitas yang disediakan pada submenu Run.

Cara membuka submenu ini ialah dengan menekan tombol Alt-R atau dengan menekan tombol kiri mouse pada Run. Fungsi Go di dalam submenu ini ialah untuk menjalankan program hingga selesai, atau hingga ditemukan break point. Jangan jalankan fungsi ini. Run to Cursor ialah fungsi untuk menjalankan program hingga di mana kursor berada pada window source. Ini sangat berguna untuk menelusuri program. Fungsi yang paling penting ialah Step Over yang dapat digantikan dengan menekan tombol F10. Gunanya ialah untuk mengeksekusi baris yang sedang ditunjuk oleh tanda panah pada window source. Dengan ini tanda panah akan melompat ke baris berikutnya. Yang perlu diingat ialah, jika ada pemanggilan subroutine, subroutine tersebut akan dijalankan hingga selesai dan tanda panah akan diletakkan pada baris setelah pemanggilan subroutine tersebut. Sehingga kita tidak dapat menelusuri jalannya program di dalam subroutine tersebut baris per baris.

Untuk menelusuri isi subroutine, kita harus menggunakan Step Into yang dapat digantikan dengan tombol F8. Dengan fungsi Step Into ini, subroutine akan dibuka dan ditelusuri satu per satu. Tetapi ingat, fungsi-fungsi seperti printf, scanf dan lain sebagainya akan ikut ditelusuri hingga sampai ke bahasa assembly-nya dengan fungsi Step Into. Untuk itu kita harus selektif dalam menentukan penggunaan Step Into dan Step Over.

Sekarang coba tutup submenu Run dan tekan F10 dua kali. Tanda panah akan berpindah ke baris:

```
Murid[a].nilai=a;
```



Gambar 2.5 Penelusuran Lebih Lanjut

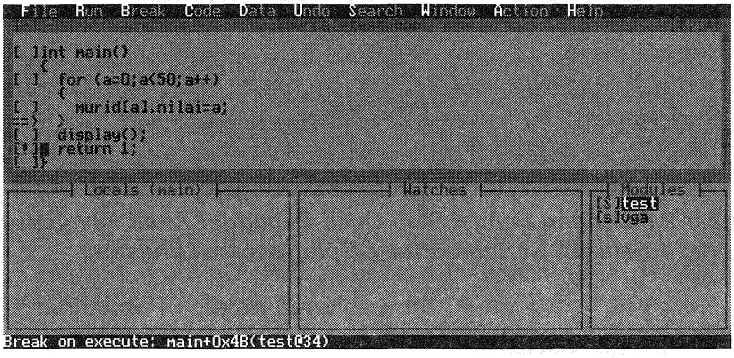
Fungsi penelusuran program ini sangat erat kaitannya dengan fungsi break point. Fungsi break point dalam Watcom Debugger ialah penandaan suatu baris sehingga pada saat program dijalankan, jika program berhenti sementara dan kembali ke layar debugger, maka pada baris yang ditandai tadi akan dieksekusi. Caranya ialah dengan menekan tombol F9 pada baris yang akan kita tandai. Baris yang berisi break point yang aktif akan ditandai dengan tanda seru (!) pada kurung di sebelah kiri baris.

Untuk menonaktifkan break point, kita cukup menekan tombol F9 sekali lagi sehingga tanda ! berubah menjadi tanda titik. Untuk melenyapkan break point, kita cukup menekan F9 sekali lagi sehingga tanda titik pada baris tersebut lenyap. Break point ini sangat penting. Misalnya jika kita hendak melacak program di atas mulai pada baris di mana dipanggilnya subroutine display, kita tidak perlu menekan F10 berkali-kali hingga sampai pada baris tersebut. Kita cukup meletakkan break point pada baris di mana dipanggil subroutine display kemudian jalankan program dengan F5. Program akan otomatis berhenti pada baris tersebut dan kembali ke debugger. Selanjutnya kita dapat meneruskan penelusuran dengan menekan tombol F8 atau F10 mulai dari baris tersebut.

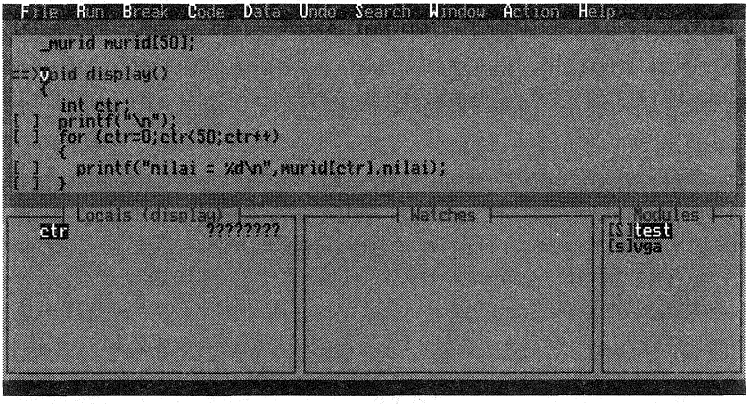
Baris yang ditandai dengan break point aktif dapat dilihat pada Gambar 2.6 di baris program return 1;.

Jika Anda ingin membuktikan perbedaan F8 dan F10, letakkan break point pada baris di mana dipanggil subroutine display, kemudian tekan F5 lalu eksekusi pemanggilan subroutine display dengan menekan F8. Debugger akan masuk ke dalam subroutine dan melakukan penelusuran di dalam subroutine tersebut. Sekarang bandingkan dengan penggunaan F10. Tekan Alt-R untuk membuka

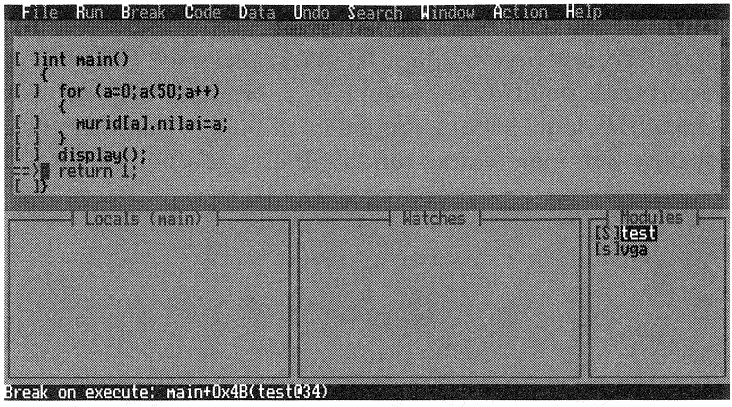
submenu Run dan pilih Restart Program untuk mengulangi proses debugging. Kemudian sekali lagi jalankan program (F5) hingga berhenti pada break point di pemanggilan subroutine display. Sekarang coba tekan F10 lagi, secara langsung debugger akan menjalankan fungsi display dan meletakkan tanda panah pada baris return 1;.



Gambar 2.6 Break Point pada Listing Program



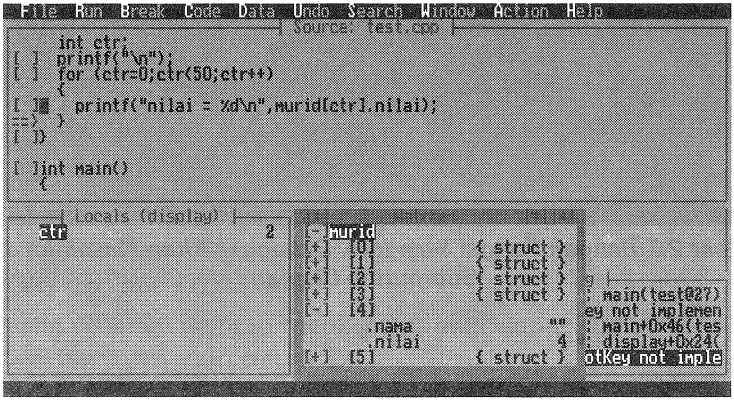
Gambar 2.7 Dengan Fungsi Step Into, Debugger Akan Masuk ke dalam Subroutine Display



Gambar 2.8 Dengan Fungsi Step Over, Debugger Akan Mengeksekusi Seluruh Isi Subroutine Display secara Langsung

Melihat Harga variabel

Watcom Debugger menyediakan fasilitas untuk melihat harga variabel (fungsi Watch). Seluruh variabel lokal akan ditampilkan secara otomatis pada window Locals (display). Coba ulangi proses debugging dengan memilih Restart Program kemudian telusuri program hingga sampai di dalam subroutine display (seperti pada Gambar 2.9).



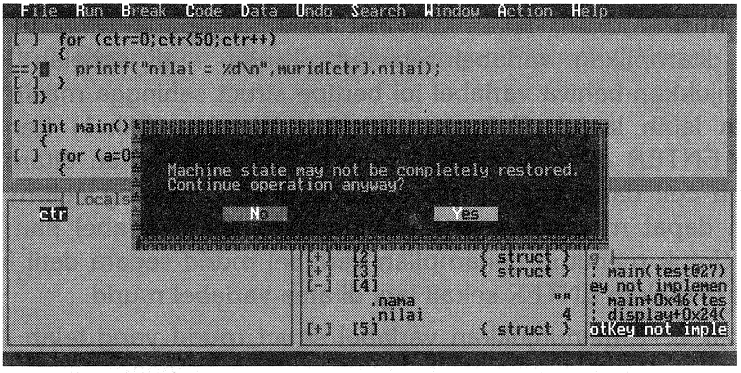
Gambar 2.9 Melihat Harga Variabel pada Window Locals (Display)

Perhatikan bahwa variabel ctr yang merupakan satu-satunya variabel lokal dalam subroutine display ditampilkan pada window Locals (display). Terus tekan tombol F10 hingga harga variabel ctr = 4, kemudian jangan kaget. Tekan tombol Ctr-Kursor kiri beberapa kali.

WOW!!! Arah penelusuran program berjalan mundur. Watcom Debugger menyediakan fasilitas rewind. Sehingga jika kita kelewatan menelusuri program, kita dapat mengulangnya lagi dengan cara berjalan mundur. Dan yang paling hebat ialah seluruh harga variabel dikembalikan ke harga pada baris yang diulang.

Untuk melanjutkan penelusuran program, kita harus menekan tombol Ctr-Kursor kanan beberapa kali hingga berada pada baris terakhir yang telah dieksekusi untuk mengembalikan kursor penelusuran pada baris terakhir kita melakukan penelusuran maju. Kemudian kita dapat melakukan penelusuran seperti biasa dengan menekan tombol F8 atau F10.

Jika Anda mendapati pesan pada Gambar 2.10, berarti kursor penelusuran masih belum berada pada lokasi yang benar, jawab No kemudian tekan Ctr-Kursor kanan beberapa kali hingga maksimal kemudian lanjutkan kegiatan penelusuran.



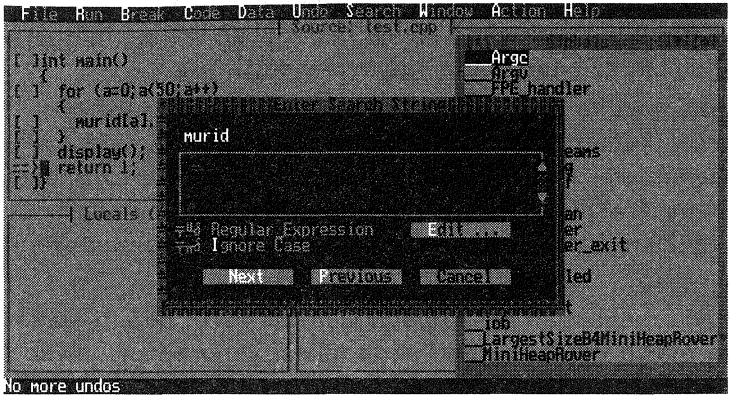
Gambar 2.10 Posisi Kursor Penelusuran Belum Berada pada Posisi yang Tepat

Melihat Harga Variabel Global

Sejauh ini kita baru melakukan pengintaian terhadap variabel lokal. Bagaimana cara melihat isi variabel global? Variabel global dapat ditampilkan pada window watches. Caranya ialah dengan membuka submenu Data (tekan Alt-D) kemudian pilih Globals.

Akan ditampilkan window Globals yang menampilkan seluruh variabel yang ada bahkan variabel yang digunakan oleh Watcom C pun ditampilkan. Untuk mencari variabel kita di antara tumpukan variabel-variabel C yang sangat banyak ini dapat digunakan fungsi pencarian string. Caranya ialah dengan menekan tombol / kemudian isi-

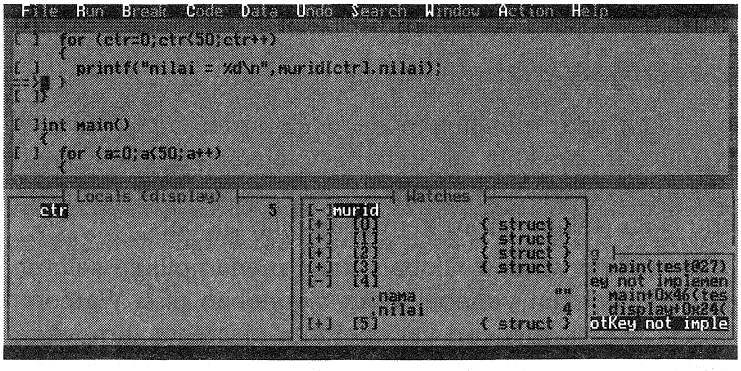
kan nama variabel yang hendak kita cari. Di sini kita akan mencari variabel murid. Seperti pada Gambar 2.11.



Gambar 2.11 Mencari Variabel Global dengan Pencarian String

Setelah ditemukan, tekan ENTER. Sekarang pada window Watch akan ditampilkan variabel tersebut. Tanda + di sebelah kirinya menunjukkan bahwa variabel ini bertipe struct sehingga masih dapat dibuka lebih lanjut. Fokuskan window Watches (Alt-1) kemudian tekan ENTER di atas variabel murid. Berikutnya akan ditampilkan seluruh cell dari variabel murid. Pada masing-masing cell-nya kita masih dapat membukanya lagi hingga melihat variabel nama dan nilai. Untuk menutup penampilan variabel murid secara detil ini kita dapat menekan ENTER sekali lagi di atas variabel murid.

Gambar 2.12 menunjukkan variabel struct murid yang telah dibuka dan cell index 4 dibuka lebih lanjut sehingga menampilkan isi dari variabel nilai dan nama.



Gambar 2.12 Membuka Isi Struct dalam Window Watches

Lain-lain

Watcom Debugger bahkan dapat membuka sebuah pointer yang menunjuk ke sebuah struct, sehingga kita dapat mengetahui isi struct tersebut. Kesulitan men-debug variabel link list pada Borland C maupun Turbo C tidak dialami dengan menggunakan Watcom Debugger, karena kita dapat menelusurinya seperti menelusuri array biasa.

Jika program yang akan kita telusuri memiliki modul yang berada pada file lain, cara membuka modul tersebut untuk ditampilkan pada window source ialah dengan menekan ENTER di atas salah satu subroutine yang berada pada modul tersebut.

Jika kita menggunakan external assembler, kita tetap masih dapat menelusuri jalannya program, bahkan kita dapat mengetahui harga setiap register dengan jalan menampilkan window register (dengan menekan tombol Alt-3).

Sampai di sinilah pembahasan kita mengenai penggunaan Watcom C utility. Selanjutnya kita akan melangkah ke babak yang paling menarik.... Pemrograman Super VGA dengan Protected Mode.

BAB 3
PENDAHULUAN PEMBAHASAN

Pada bab-bab selanjutnya kita akan mulai melakukan pembahasan materi pokok dari buku ini yaitu Pemrograman Super VGA 8, 16, 24 dan 32 bit. Di bab ini kita akan membahas sedikit mengenai di mana para pembaca dapat mencari listing program dan utility-utility beserta informasi cara pemakaiannya yang telah disediakan seiring dengan pembahasan materi dalam buku ini.

3.1 Utility Pembantu

Program-program sebagai utility pembantu disediakan pada directory TOOL. Adapun utility-utility ini adalah sebagai berikut:

CAP2.EXE

Utility ini digunakan untuk meng-capture (menangkap) gambar pada layar monitor dengan mode graphic 320 x 200 256 warna. Caranya ialah dengan memanggil CAP2 beserta parameter directory tempat hasil capture akan disimpan. Tombol untuk mengaktifkan capture ialah Shift-PrintScreen.

Misalnya kita hendak meletakkan hasil capture pada directory C:\PICTURE maka kita harus memanggil:

```
CAP2 C:\PICTURE\
```

File akan otomatis dinamakan 1.PIC pada proses capture pertama kali, berikutnya angka ini akan terus bertambah menjadi 2.PIC, 3.PIC dan seterusnya. Format file gambar ialah format memory alamat $A0000 dengan jumlah titik 320 x 200.

CAPPAL.EXE

Utility ini digunakan untuk meng-capture tabel palette pada layar monitor dengan mode graphic 320 x 200 256 warna. Caranya ialah dengan memanggil CAPPAL beserta parameter directory tempat hasil capture akan disimpan. Tombol untuk mengaktifkan capture ialah Shift-PrintScreen.

Misalnya kita hendak meletakkan hasil capture pada directory C:\PICTURE maka kita harus memanggil:

```
CAPPAL C:\PICTURE\
```

File akan otomatis dinamakan 1.PAL pada proses capture pertama kali, berikutnya angka ini akan terus bertambah menjadi 2.PAL, 3.PAL dan seterusnya. Format file palette ini sama dengan tabel yang akan kita bahas pada bab mengenai perakitan graphic library 8 bit tentang file VGAPAL.CPP.

PCXCON8.EXE

Utility ini digunakan untuk mengubah format gambar dari PCX 8 bit ke SCR 8 bit untuk digunakan sebagai background dari fungsi parallax scroll. Caranya ialah dengan memanggil PCXCON8 beserta parameter nama file yang akan diubah lengkap beserta ekstensinya.

Misalnya kita hendak mengubah file BACK1.PCX maka kita harus memanggil:

```
PCXCON8 BACK1.PCX
```

File output akan dinamakan BK1.SCR dan BK1.ATR secara otomatis.

Ketiga listing program dari utility di atas juga disediakan dalam bahasa Pascal jika para pembaca ingin memodifikasi dan mempelajarinya.

3.2 Listing Program dan Executable File

Untuk memudahkan pembahasan, karena nantinya kita akan membagi topik pembicaraan dalam dua bagian yang berbeda yaitu mode graphic 8 bit dan mode graphic 24 bit maka akan disediakan dua buah directory yaitu 8BIT dan 24BIT. Dalam kedua directory ini dibagi lagi menjadi dua buah directory yaitu EXE dan SOURCE. Directory SOURCE menampung segala listing program, sedangkan directory EXE menampung program hasil compile yang sudah berupa executable file beserta file-file gambar dan palette. Jika para pembaca ingin mencari file yang bersangkutan dengan mode 8 bit maka para pembaca dapat mencarinya pada directory 8BIT. Demikian pula untuk 24 bit.

Dalam kedua directory EXE dari dua mode ini disediakan file DOS4GW.EXE sehingga file-file executable tersebut dapat dijalankan langsung pada directory tersebut.

3.3 Hal-hal yang Perlu Diperhatikan

Kemampuan Memory

Jika beberapa program tidak jalan, cobalah melihat jumlah memory sistem komputer Anda yang tersisa. Karena kita menggunakan mode graphic dengan resolusi tinggi sehingga membutuhkan banyak memory. Untuk menjalankan semua program yang kita bahas di sini kita membutuhkan memory sisa sebesar sekitar 2 Mega Byte. Jika memory telah mencukupi dan program tetap belum jalan, matikan WINDOWS dan boot sistem komputer Anda ke DOS. Juga matikan segala memory manager yang Anda pakai.

Kemampuan Komputer

Untuk menjalankan program-program hasil compile dari Watcom, dibutuhkan paling sedikit komputer 386 karena dibutuhkannya mode protected mode. Proses compile pada komputer di bawah pentium memang agak lambat. Hal ini karena Watcom mengadakan optimasi besar-besaran terhadap listing program. Jadi jangan menganggap komputer Anda macet jika proses compile terlihat sangat lama. Proses selama 5 menit termasuk masih masuk akal untuk program yang cukup panjang.

Pengaturan AUTOEXEC dan CONFIG.SYS

Perhatikan kedua file ini karena seluruh PATH dari Watcom C harus diset agar kita dapat menjalankan semua utility Watcom C dengan benar.

BAB 4

DASAR PEMROGRAMAN SUPER VGA

Pada saat mulai bermunculannya monitor-monitor beserta card-card Super VGA, mulailah tampilan software komputer terlihat jauh lebih realistis karena didukung dengan resolusi yang tinggi seperti 800 x 600, 1024 x 768 dan sebagainya. Karena card-card Super VGA ini dirancang oleh perusahaan yang berbeda, maka mulai timbullah masalah kompatibilitas antara software dengan card-card ini.

Untuk itu para software programmer menyediakan bermacam-macam driver yang masing-masing mendukung setiap card Super VGA yang berbeda. Hal ini tentunya sangat tidak efisien karena selain waktu pemrograman lebih lama, juga ukuran dari software membengkak. Lebih-lebih jika ada beberapa card Super VGA yang tidak didukung, para pengguna card Super VGA ini akan merasa dirugikan.

Sebagai pemecahan, disepakati sebuah interface standar yang harus diikuti oleh setiap perusahaan VGA card yang dinamakan VESA. Dengan ini, cara pemrograman setiap card Super VGA tidak berbeda satu dengan yang lainnya.

Hampir seluruh Super VGA card yang sekarang beredar di pasaran mengikuti standar VESA. Juga pembahasan kita pada buku ini akan

mengikuti standar VESA. Walaupun resolusi yang didukung oleh standar VESA ada bermacam-macam, kebanyakan contoh yang akan diulas dirakit dengan menggunakan mode resolusi 640 x 400 dengan tujuan meminimalkan jumlah memory VGA card yang dibutuhkan bagi para pembaca sehingga para pembaca yang memiliki VGA card bermemory rendah masih dapat mengikuti.

4.1 Meneliti Card Super VGA

Super VGA card ada bermacam-macam jenis, seperti Tseng, Diamond, Matrox Millenium dan lain sebagainya. Di dalam card Super VGA, terdapat memory seperti pada memory komputer. Ukuran memory ini menentukan resolusi dan warna maksimum yang dapat ditampilkan oleh card tersebut.

Untuk itu kita dapat melihat buku manual saat membeli card Super VGA tersebut.

Adapun dengan perkiraan cara mengukur resolusi dan jumlah warna yang dapat ditampilkan oleh sebuah card Super VGA ialah dengan menghitung kapasitas memory yang diperlukan oleh suatu mode graphics.

Misalnya kita hendak menghitung kapasitas memory yang diperlukan untuk mode graphics 640 x 400 8 bit warna.

Kapasitas memory = 640 x 400 x 1 byte = 256.000 bytes.

Jika memory card Super VGA Anda hanya 256 kbytes maka mode graphic 640 x 400 8 bit warna di ataslah mode graphic maksimum yang dapat ditampilkan.

Jika kapasitas memory suatu card = 1 Mbyte maka mode graphics yang didukung oleh card tersebut secara perkiraan ialah seperti pada Tabel 4.1.

Tabel 4.1 Perkiraan Mode Graphic yang Dapat Ditampilkan dengan Memory 1 Mbyte

Resolusi	Jumlah Bit Per Titik	Jumlah Warna	Keperluan Memory
320 x 200	8	256	64.000 Bytes
640 x 400	8	256	256.000 Bytes
640 x 480	8	256	307.200 Bytes

8 00 x 600	8	256	480.000 Bytes
1024 x 768	8	256	786.432 Bytes
320 x 200	16	64.000	128.000 Bytes
640 x 400	16	64.000	512.000 Bytes
640 x 480	16	64.000	614.400 Bytes
800 x 600	16	64.000	960.000 Bytes
320 x 200	24	16 Juta	192.000 Bytes
640 x 400	24	16 Juta	768.000 Bytes
640 x 480	24	16 Juta	921.600 Bytes
320 x 200	32	1024 Juta	256.000 Bytes
640 x 400	32	1024 Juta	1024.000 Bytes

Perkiraan di atas tidak selalu benar, karena setiap Super VGA card memiliki karakteristik yang berbeda. Ada Super VGA card yang tidak mendukung mode graphic 320 x 200 24 bit walaupun memory yang dimiliki lebih dari 1 Mbyte. Untuk itu kita harus selalu berpedoman pada manual Super VGA card kita.

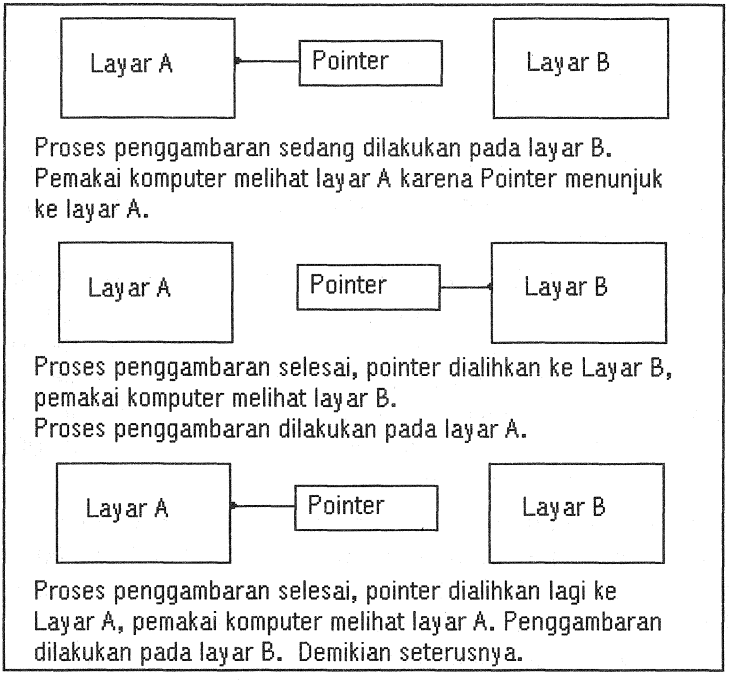
Hampir seluruh Super VGA card menyediakan fasilitas Page flipping. Fasilitas ini memungkinkan kita melakukan animasi dua layar (double buffering animation) seperti yang telah kita lakukan pada pembahasan di seri-seri sebelumnya.

Catatan

Animasi dua layar ialah memanfaatkan layar buatan (virtual screen) untuk melakukan seluruh proses penggambaran. (Pada saat ini pemakai komputer masih melihat layar tampilan.) Setelah proses penggambaran selesai, layar buatan ini dipindahkan ke layar tampilan. Dengan cara ini, pemakai komputer tidak melihat jalannya proses penggambaran yang kadang menimbulkan efek berkedip-kedip.

Tentu saja Page flipping di sini tidak perlu menggunakan memory RAM komputer sebagai layar buatan, melainkan menggunakan memory di card Super VGA. Untuk memindahkan layar buatan ini ke layar tampilan agar dilihat oleh pemakai komputer, kita tidak perlu melakukan transfer data secara manual, melainkan cukup memindah pointer dari Super VGA ke layar buatan. Sehingga sekarang layar buatanlah yang menjadi layar tampilan.

Proses ini dapat digambarkan pada Gambar 4.1. Layar A dan Layar B berada dalam memory card Super VGA. Pointer super VGA card menunjuk salah satu layar yang akan ditampilkan pada monitor.



Gambar 4.1 Proses Penggunaan Page Flipping

Terkadang ada beberapa card Super VGA yang mendukung hingga 4 Layar. Dengan ini, pemakaian memory pada mode graphic tersebut menjadi 4 kali lebih besar.

Karena page flipping ini tidak selalu sama pada setiap card Super Vga, maka pembahasan kita tidak akan menggunakan hardware page flipping (page flipping dengan memindahkan pointer seperti yang dijelaskan di atas) melainkan secara manual dengan transfer data.

Mode Graphic dengan jumlah bit per pixel 32 bit hanya didukung oleh card-card Super VGA yang baru.

4.2 Mengakses Card Super VGA

Untuk mengalokasikan titik pada layar VGA, digunakan memory segment $A000 mulai offset 0 hingga offset $FFFF. Titik-titik yang

ditampilkan pada layar monitor sebenarnya berada pada alamat memory ini. Kalau kita hendak membaca ataupun mengubah sebuah titik di layar monitor, kita cukup membaca atau mengganti salah satu harga pada alamat memory tersebut. Untuk mode graphic 320 x 200 256 warna tentu saja tidak mengalami kesulitan karena jumlah memory yang diperlukan ialah 64000 bytes dan kebetulan besar memory yang disediakan pada segment $A000 ialah 64Kbytes = 65536 Bytes.

Dengan ini, titik-titik pada seluruh layar monitor dapat dicerminkan pada alamat memory segment $A000 tersebut dengan urutan seperti yang dijelaskan pada Gambar 4.2.

Titik pada koordinat (0,0) berada pada alamat memory $A000:0000, titik pada koordinat (1,0) berada pada alamat memory $A000:0001, titik pada koordinat (1,1) berada pada alamat memory $A000:0321 dan titik pada koordinat (319,199) berada pada alamat memory $A000:63999. Karena titik ini adalah titik terakhir pada mode graphic 320 x 200 256 warna, maka sisa memory mulai $A000:64000 hingga $A000:65535 tidak dipergunakan.

Nah, sekarang bagaimana dengan mode graphic yang lain seperti 640 x 400 256 warna? Jumlah memory yang dibutuhkan ialah 640 x 400 x 1 byte = 256000 byte, sedangkan jumlah memory yang disediakan pada segment $A000 hanya 65536 byte. Berarti hanya seperempat layar saja yang cukup ditampung dalam alamat memory ini? Benar! Memang hanya seperempat layar saja yang ditampung dalam alamat memory ini pada satu saat. Dengan kata lain, alamat memory segment $A000 ini hanya merupakan jendela (viewport) yang mewakili bagian dari layar monitor. Untuk mengakses bagian yang tidak berada di dalam jendela yang sedang mewakili bagian dari layar monitor, kita harus memindahkan jendela tersebut dengan menggunakan interrupt DOS.

Gambaran struktur pembagian dari sistem jendela (viewport) ini dapat kita lihat pada Gambar 4.3.

Terlihat bahwa pada mode graphic 640 x 400 256 warna, ada 4 buah jendela yang digunakan. Ketiga jendela pertama berukuran 65536 Bytes, karena ini adalah jumlah maksimal yang dapat diakses secara langsung lewat memory segment $A000. Pada jendela terakhir, tersisa jumlah titik sebanyak 59392 byte (1 byte = 1 titik) = 59392 titik.

Dengan adanya pembagian memory seperti ini, jika yang sedang diaktifkan ialah jendela nomor 1 maka titik yang dapat diakses ialah titik mulai koordinat (0,0) hingga titik pada koordinat (255,102)

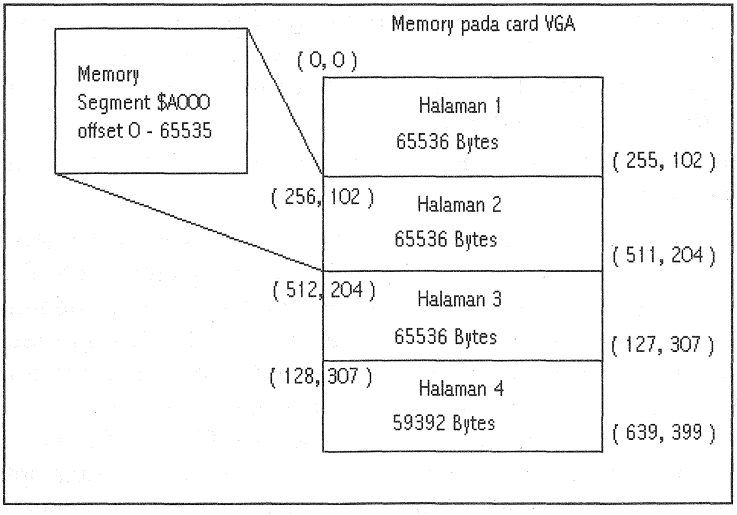
karena titik pada koordinat (255,102) ini telah berada pada offset maksimal yaitu 65535 pada segment $A000.

Segment $A000

0	1	2		318	319
320	321	322		638	639
63680	63681	63682		63998	63999

Gambar 4.2 Sistem Alokasi Titik pada Mode Graphic 320 X 200 256 Warna

Untuk mengakses titik pada koordinat selanjutnya, kita harus mengaktifkan jendela yang bersangkutan dengan interrupt DOS.



Gambar 4.3 Struktur Memory Card VGA pada Mode Graphic 640 X 400 256 Warna

Untuk mode graphic 640 x 480 256 warna, digunakan 5 buah jendela. Mode graphic 800 x 600 256 warna menggunakan 8 buah jendela. Cara menghitungnya secara cepat tanpa perlu menggambar diagram seperti Gambar 4.3 ialah dengan membagi jumlah keperluan memory suatu mode graphic dengan ukuran memory segment $A000 yaitu 65536. Hal ini juga berlaku untuk mode graphic 16 bit, 24 bit maupun 32 bit.

Adapun secara garis besar, alokasi titik pada layar VGA dibedakan menjadi dua macam berdasarkan mode graphic yang dipilih. Jenis yang pertama ialah menggunakan tabel warna dan jenis yang kedua tanpa menggunakan tabel warna (color look up table).

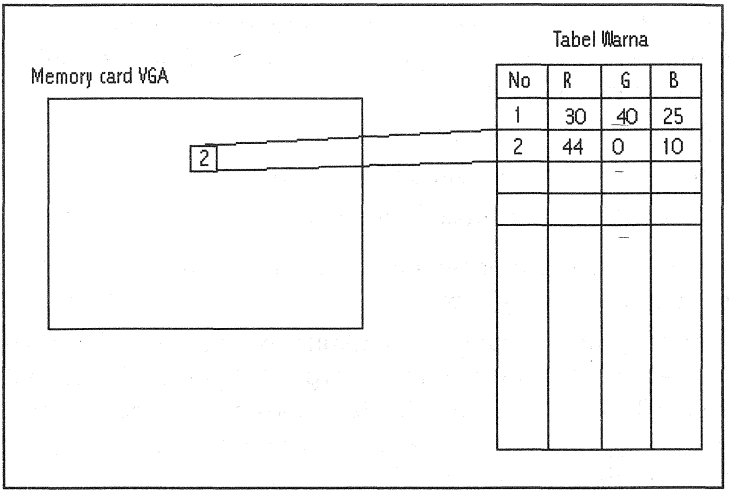
Jenis yang pertama ini digunakan oleh mode graphic yang memiliki 256 warna, sedangkan jenis kedua digunakan oleh mode graphic di atas 256 warna (16 bit, 24 bit dan 32 bit).

4.3 Alokasi Titik Menggunakan Tabel Warna

Alokasi titik jenis ini selalu kita pakai pada seri-seri Pemrograman Animasi dan Game Profesional sebelumnya. Pada jenis ini, titik yang disimpan dalam memory segment $A000 ialah indeks sebuah tabel warna yang memiliki 256 pilihan warna. Karena adanya 256 pilihan warna, harga indeks ini berkisar antara 0 hingga 255. Jadi jika titik yang kita letakkan pada layar berharga 30, ini berarti bukan titik berwarna 30 yang diletakkan di layar melainkan titik yang memiliki warna yang didefinisikan pada tabel warna nomor indeks 30 yang sebenarnya diletakkan di layar.

Sistem alokasi titik menggunakan tabel warna ini dapat kita lihat lebih jelas pada Gambar 3.5.

Di sana ditunjukkan sebuah titik menggunakan warna dengan nomor index 2. Pada tabel warna, terdapat tiga buah variabel yaitu R (warna merah), G (warna hijau) dan B (warna biru) yang menunjukkan campuran warna pembentuk warna nomor 2. Masing-masingnya memiliki harga berkisar antara 0 hingga 63. Warna hijau terang ialah R = 0, G = 63 dan B = 0. Warna merah terang ialah R = 63, G = 0 dan B = 0 sedangkan warna putih terang ialah R = 63, G = 63 dan B = 63. Tabel warna ini dapat diubah dengan menggunakan interrupt DOS. Adapun tekniknya tidak akan kita bahas secara mendetil karena telah diberikan pada seri terdahulu.



Gambar 4.4 Alokasi Titik dengan Tabel Warna

4.4 Alokasi Titik Tanpa Tabel Warna

Sistem ini mulai digunakan dengan munculnya card Super VGA yang mendukung jumlah warna di atas 256 warna. Dengan jumlah warna 64000 ke atas, maka sistem penggunaan tabel tidak lagi efisien karena ukuran tabel tentu saja harus sesuai dengan jumlah warna yang dapat ditampilkan. Untuk itu pada setiap titik, data yang disimpan bukanlah indeks tabel melainkan langsung nilai R, G dan B dari warna yang dipergunakan oleh titik tersebut.

Pada mode graphic yang mendukung 16 bit per titik (64000 warna), setiap titiknya memiliki harga 2 byte sehingga titik pada koordinat (0,0) berada pada jendela nomor 1 dengan alamat offset nomor 0 dan nomor 1 (menempati dua cell memory). Dua byte ini memiliki pembagian sebagai berikut:

Bit 0-4 Intensitas Biru (Unsur B).

Bit 5-9 Intensitas Green (Unsur G).

Bit 10-14 Intensitas Red (Unsur R).

Bit 15 Tidak terpakai.

Dengan ini, masing-masing intensitas memiliki resolusi 5 bit yaitu berkisar antara 0 hingga 31.

Pada mode graphic yang mendukung 24 bit per titik (16 juta warna), setiap titiknya memiliki harga 3 byte sehingga titik pada koordinat

(0, 0) berada pada jendela nomor 1 dengan alamat offset nomor 0, 1 dan nomor 2 (menempati tiga cell memory). Tiga byte ini memiliki pembagian sebagai berikut:

Bit 0-7	Intensitas Biru (Unsur B).
Bit 8-15	Intensitas Green (Unsur G).
Bit 16-23	Intensitas Red (Unsur R).

Dengan ini, masing-masing intensitas memiliki resolusi 8 bit yaitu berkisar antara 0 hingga 255.

Pada mode graphic yang mendukung 32 bit per titik (1024 juta warna), setiap titiknya memiliki harga 4 byte sehingga titik pada koordinat (0,0) berada pada jendela nomor 1 dengan alamat offset nomor 0 hingga nomor 3 (menempati empat cell memory). Empat byte ini memiliki pembagian sebagai berikut:

Bit 0-9	Intensitas Biru (Unsur B).
Bit 10-19	Intensitas Green (Unsur G).
Bit 20-29	Intensitas Red (Unsur R).
Bit 30-31	Tidak terpakai.

Dengan ini, masing-masing intensitas memiliki resolusi 10 bit yaitu berkisar antara 0 hingga 1023.

Akhir-akhir ini mulai muncul 64 bit card VGA yang tentu saja memiliki resolusi warna yang lebih halus dibandingkan dengan ketiga mode graphic di atas. Pembahasan 64 bit card ini tidak akan dilakukan pada edisi kali ini. Kita hanya akan membahas 8, 16, 24 dan 32 bit warna.

4.5 Penggunaan Layar Buatan (Virtual Screen)

Tentu saja para pembaca sekalian mulai bertanya-tanya, bagaimana cara membuat sebuah graphic library standard yang dapat mendukung seluruh mode graphic. Jawabannya hampir tidak mungkin. Kita harus membuat graphic library khusus untuk mode 8 bit warna, khusus untuk 16 bit, 24 bit dan 32 bit. Sebenarnya kita dapat hanya mengganti fungsi PutPixel untuk setiap mode bit warna dan selalu menggunakan fungsi PutPixel ini untuk meletakkan gambar, menscroll gambar, membuat bangun-bangun geometri. Tetapi dalam pembuatan game, hal ini tidak dianjurkan karena kecepatan program akan menjadi sangat lambat dengan perhitungan alamat memory pada PutPixel yang terlalu sering.

Untuk itu pada pembahasan kita kali ini, kita akan membahas sebuah graphic library yang mendukung resolusi 320 x 200, 640 x 400, 640 x 480 dan 1024 x 768 dengan warna 8 bit yang akan dirakit dengan assembler penuh untuk memaksimalkan kecepatan. Selanjutnya kita akan meneruskan pembahasan untuk graphic library yang menggunakan gabungan antara assembler dan C untuk mode warna 16 bit, 24 bit dan 32 bit. Ketiga mode bit warna ini akan digabung dalam satu graphic library yang tentu saja memiliki kecepatan yang lambat. Untuk optimasi kecepatan, pembaca dapat memisahkan ketiga bit warna ini menjadi tiga graphic library yang berbeda dengan contoh yang telah diberikan pada pembuatan graphic library 8 bit sebelumnya. (Hitung-hitung sebagai latihan bagi Anda dan sekaligus mengurangi beban bagi pengarang ☺.)

BAB 5

PERAKITAN GRAPHICS LIBRARY 8 BIT PERTAMA

Perakitan graphics library 8 bit (256 warna) ini akan dilakukan sepenuhnya dalam bahasa assembler yang di-compile secara external. Hal ini karena inline assembler dari compiler Watcom C kurang lengkap dibanding dengan TASM.EXE (Turbo Assembler) compiler. Untuk itu dalam proses compile graphics library ini, kita akan menggunakan Turbo Assembler compiler seperti yang telah disebutkan di atas. Utility ini dapat Anda temukan di paket Borland C ataupun Turbo C.

Adapun cara pemanggilan TASM.EXE beserta parameternya ialah sebagai berikut:

```
TASM /os /zi fileinput fileoutput
```

Parameter /os berarti proses compile dilakukan untuk menciptakan object file bertipe standar.

Parameter /zi berarti diikutsertakannya informasi debug secara lengkap.

Fileinput ialah nama file yang akan di-compile beserta ekstensinya. (.ASM).

Fileoutput ialah nama file yang akan diciptakan beserta ekstensinya (.OBJ).

Library yang akan kita bahas ini dapat Anda temukan pada directory source dengan nama VGA.ASM. Adapun file header-nya yang berisi prototype dari setiap function dan berfungsi sebagai penghubung ke compiler C bernama TRY.H.

5.1 Pemrograman Assembly dengan Register 32 bit

Dalam 32 bit programming, dikenal register-register tambahan yaitu eax, ebx, ecx, edx yang merupakan 16 bit tambahan pada register ax, bx, cx dan dx sehingga masing-masing merupakan register 32 bit. Hal ini sama seperti ax, bx, cx dan dx yang merupakan 8 bit tambahan dari register al, bl, cl dan dl. Register-register yang lain seperti es, ds, bp, sp pun ditambah menjadi 32 bit sebagai register esi, edi, ebp dan esp.

Dengan ini yang biasanya kita menggunakan kombinasi es:di dan ds:si (Real mode) untuk transfer data, sekarang kita cukup menggunakan satu register esi dan edi (Protected mode).

Catatan

Dalam pemrograman 32 bit, sejauh mungkin hindarilah penggunaan operasi terhadap register / data 8 bit maupun 16 bit karena prosesnya jauh lebih pelan dibandingkan dengan operasi register / data 32 bit. Aneh khan? Hal ini karena komputer yang mendukung pemrograman 32 bit dirancang khusus untuk melakukan operasi 32 bit secara cepat, tetapi tidak dalam operasi 16 maupun 8 bit.

Yang dimaksud dengan operasi 8 bit, 16 bit dan 32 bit ini ialah seperti contoh berikut:

```
mov  al,bl      ; operasi 8 bit
mov  ax,bx      ; operasi 16 bit
mov  eax,ebx    ; operasi 32 bit
```

Model Program VGA.ASM

```
; ------------------------------------------------------------
; Nama       : VGA.ASM
; Definisi   : Graphics library 8 bit.
;                Mode graphic yang didukung:
;                     320 x 200  256 warna
;                     640 x 400  256 warna
```

```
;                    800 x 600   256 warna
;                   1024 x 768   256 warna
; Programmer: Agustinus Nalwan
; ---------------------------------------------------------

.386P
.MODEL FLAT
```

Jenis program ialah 386 dengan memory model FLAT yang berarti digunakannya protected mode (tanpa segment-offset).

Struktur Data

```
.DATA
        V_X1    DD      ?
        V_Y1    DD      ?
        V_X2    DD      ?
        V_Y2    DD      ?
        S_L     DD      ?
        S_W     DD      ?
        A_SL    DD      ?
        A_SW    DD      ?
        W_Scr   DD      ?
        W_Loop  DD      ?
        W_Acc   DD      ?

_TEXT SEGMENT DWORD PUBLIC 'CODE'
        ASSUME cs:_TEXT, ds:_DATA
```

Struktur data di atas ini digunakan untuk menampung informasi-informasi yang menjadi karakteristik setiap mode graphic, seperti jumlah jendela, jumlah byte pada halaman terakhir, batas maksimum koordinat X dan Y.

Secara lengkapnya ialah sebagai berikut:

V_X1, V_Y1 ialah koordinat kiri atas viewport.

V_X2, V_Y2 ialah koordinat kanan bawah viewport.

S_L ialah koordinat X maksimum.

S_W ialah koordinat Y maksimum.

A_SL ialah jumlah titik horizontal.

A_SW ialah jumlah titik vertikal.

W_Scr ialah jumlah memory yang diperlukan dalam satu layar dalam satuan 32-bit (1 unit W_Scr = 4 byte).

W_Loop ialah jumlah (jendela-1) yang diperlukan.

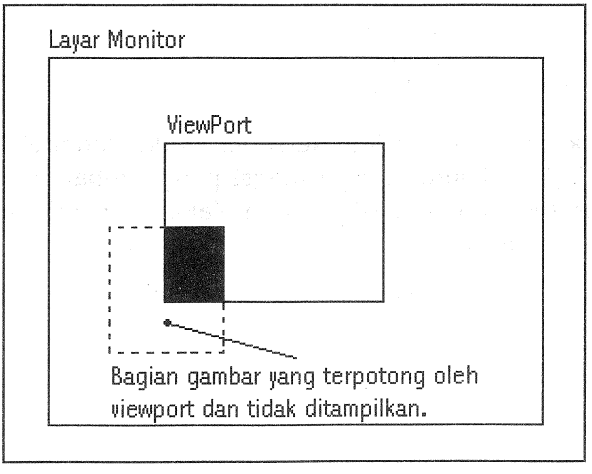
W_Acc ialah jumlah sisa byte pada jendela terakhir (juga dalam satuan 32-bit).

Dengan digunakannya variabel-variabel ini, maka graphic library ini akan dapat mendukung mode graphic yang bermacam-macam dengan cukup mengganti harga-harga variabel di atas. (Tetapi tetap hanya untuk mode-mode graphic 8 bit.)

5.2 Sistem Viewport

Lain dengan unit VGACAD.TPU yang dulu kita bahas, unit VGA.ASM ini dirancang dengan menggunakan sistem viewport. Bagi para pembaca yang telah berpengalaman menggunakan GRAPH.TPU buatan Borland tentu telah mengetahui tentang viewport ini.

Dengan teknik viewport, kita dapat menentukan sebuah bingkai/persegi panjang pada layar monitor di mana hanya bagian dalam bingkai saja yang dapat digambari. Jika kita melakukan penggambaran yang melebihi bingkai tersebut, maka bagian yang melebihi tersebut akan dipotong. Dengan ini kita dapat menciptakan window-window buatan pada layar monitor. Gambar 5.1 menggambarkan sistem viewport.



Gambar 5.1 Sebuah Gambar yang Terpotong Oleh Viewport

Adapun implementasi dari teknik viewport ini sebenarnya sangat sederhana, yaitu dengan menentukan koordinat kiri atas dan kanan

bawah viewport. Kemudian seluruh proses penggambaran pada layar monitor akan dipotong berdasarkan koordinat kiri atas dan kanan bawah dari viewport ini, sama seperti teknik memotong gambar yang menembus batasan layar.

Dalam VGA.ASM, hampir seluruh fungsi dirancang dalam dua macam. Yang memiliki unsur kata VWP berarti akan dipotong berdasarkan viewport, sedangkan yang tidak memiliki awalan ini tidak akan mempedulikan keberadaan viewport. Misalnya seperti fungsi VESA_Vline dan VESA_VWP_Vline.

Fungsi ViewPort

Fungsi ini digunakan untuk mengatur ukuran viewport.

```
ViewPort    PROC NEAR
        push    ebp
        mov     ebp,esp
        ARG     VX1:DWORD,VY1:DWORD,VX2:DWORD,VY2:DWORD
```

Parameter yang diterima ialah koordinat kiri atas viewport (VX1,VY1) dan koordinat kanan bawah viewport (VX2,VY2).

```
        mov     eax,[VX1]
        mov     [V_X1],eax
        mov     eax,[VY1]
        mov     [V_Y1],eax
        mov     eax,[VX2]
        mov     [V_X2],eax
        mov     eax,[VY2]
        mov     [V_Y2],eax
```

Masukkan koordinat-koordinat tersebut pada variabel V_X1, V_Y1 dan V_X2, V_Y2. Seluruh fungsi-fungsi penggambaran yang mendukung viewport secara otomatis akan melakukan pembatasan gambar pada koordinat viewport ini.

```
        mov     esp,ebp
        pop     ebp
        ret

ViewPort    ENDP
```

5.3 Fungsi-fungsi Standar

Yang digolongkan ke dalam fungsi-fungsi standar di sini ialah fungsi untuk membuka dan menutup mode graphic, juga fungsi-fungsi ani-

masi dua layar yang bertugas untuk melakukan transfer data dari satu layar ke layar lainnya.

Seperti yang telah kita bahas pada bab sebelumnya, untuk meletakkan titik pada koordinat tertentu, kita perlu menentukan jendela di mana titik tersebut berada. Jika kita hendak meletakkan sesuatu yang bukan hanya sebuah titik, misalnya sebuah gambar, gambar ini dapat menempati beberapa jendela sekaligus. Tentu saja pada saat kita meletakkan gambar ini, kita harus selalu mengubah jendela yang aktif sesuai dengan koordinat titik yang sekarang sedang kita alokasikan. Hal ini tentu menyebabkan perhitungan yang terlalu rumit. Apalagi jika kita hendak meletakkan beberapa puluh gambar. Untuk setiap gambar kita harus melompat-lompat dari jendela yang satu ke jendela yang lainnya. Untuk itu digunakan layar buatan sebagai pemecahan.

Segala penggambaran pada layar monitor harus dilakukan pada layar buatan (virtual screen) yang berada pada RAM. Setelah proses penggambaran selesai, maka virtual screen ini akan langsung ditransfer pada masing-masing jendela di memory card VGA. Hal ini sama persis dengan teknik animasi dua/tiga layar yang telah kita pelajari pada buku SAP Pemrograman Animasi dan Game Profesional yang pertama.

Dengan ini, kita tidak dapat secara langsung melakukan penggambaran pada layar monitor seperti halnya pada unit VGACAD.TPU terdahulu dengan langsung menggunakan memory segment $A000 sebagai SOURCE dalam setiap penggambaran ke layar. Hal ini bukanlah menjadi masalah, karena kita memang selalu akan menggunakan teknik animasi dua layar ini selama pemrograman grafik, sehingga kita tidak perlu menggambar langsung pada layar monitor.

Adapun beberapa fungsi seperti pemindahan virtual screen ke virtual screen, pemindahan virtual screen ke layar monitor dan penghapusan virtual screen masing-masing diciptakan dalam dua macam. Yang pertama ialah untuk mode resolusi 320 x 200 256 warna, yaitu bernama MoveDouble, MoveToScreen dan ClearDouble. Sedangkan yang kedua ialah untuk mode resolusi 640 x 400, 640 x 480, 800 x 600 dan 1024 x 768 256 warna dengan nama VESA_MoveDouble, VESA_MoveToScreen dan VESA_ClearDouble. Hal ini ditujukan untuk mempercepat proses transfer data karena jika kita tinjau kembali, pada mode graphic 320 x 200 256 warna, tidak diperlukan sistem penggantian jendela sedangkan pada mode graphic yang lain diperlukan penggantian jendela karena ukuran memory yang diperlukan melebihi 64 Kbytes.

Fungsi MoveDouble

Fungsi ini digunakan untuk memindahkan data dari satu layar ke layar lain sejumlah 64000 byte. Seperti yang telah kita bahas di atas, fungsi ini hanya digunakan untuk memindahkan layar dengan mode resolusi 320 x 200 256 warna.

```
MoveDouble      PROC NEAR
        push    ebp
        mov     ebp, esp
        ARG     SOURCE:DWORD, DEST:DWORD
```

Parameter yang dikirimkan ialah SOURCE = alamat memory di mana data akan dipindahkan dan DEST = alamat memory tempat data akan diletakkan.

```
        mov     esi, [SOURCE]
        mov     edi, [DEST]
```

Arahkan esi dan edi ke SOURCE dan DEST.

```
        mov     ecx, 16000
```

Set counter sejumlah 16000 kali (1 unit = 32 bit) sehingga 16000 = 64000 bytes.

Dipilih mode transfer data 32 bit karena prosesnya jauh lebih cepat dibandingkan transfer data 8 atau 16 bit.

```
        rep     movsd
```

Transfer data dari SOURCE ke DEST.

```
        mov     esp, ebp
        pop     ebp
        ret

MoveDouble      ENDP
```

Fungsi MoveToScreen

Fungsi ini digunakan untuk memindahkan virtual screen ke layar monitor. Hanya digunakan untuk mode resolusi 320 x 200 256 warna.

```
MoveToScreen    PROC NEAR
        push    ebp
        mov     ebp, esp
        ARG     SOURCE:DWORD
```

Parameter yang diterima ialah SOURCE yang berisi alamat memory dari virtual screen.

```
        mov     esi, [SOURCE]
        mov     edi, 0A0000h
```

Arahkan target ke memory segment $A000 (dalam protected mode tidak dikenal segment sehingga alamat memory langsung ditulis dalam satuan byte. Segment $A000 = $A0000 bytes karena satu unit dalam segment = 16 byte = $10 hexa).

```
        mov     ecx, 16000
        rep     movsd
```

Transfer data dari virtual screen ke layar sejumlah 16000 x 4 byte = 64000 byte.

```
        mov     esp, ebp
        pop     ebp
        ret

MoveToScreen    ENDP
```

Fungsi ClearDouble

Fungsi ini digunakan untuk menghapus virtual screen dengan mode graphic 320 x 200 256 warna.

```
ClearDouble     PROC NEAR
        push    ebp
        mov     ebp,esp
        ARG     SOURCE:DWORD,COL:BYTE
```

Parameter yang diterima ialah SOURCE yaitu alamat virtual screen yang akan dihapus dan COL yang berisi warna yang digunakan untuk menghapus.

```
        mov     edi, [SOURCE]
        mov     al,[COL]
        mov     ah,[COL]
```

Arahkan target register ke SOURCE dan isikan warna pada register al dan ah.

```
        xor     edx,edx
        mov     dx,ax
        shl     eax,16
        add     eax,edx
```

Proses di atas dilakukan untuk mengisi 4 byte dalam register eax dengan harga COL. Caranya ialah dengan menyimpan register ax yang telah berisi harga COL di byte pertama dan keduanya dalam register dx. Kemudian geser eax 16 bit ke kiri sehingga harga COL di dalam register eax sekarang berada pada byte ketiga dan keempat. Langkah selanjutnya ialah mengisi byte pertama dan kedua dengan harga COL yaitu dengan menambahkan register edx yang memiliki harga COL pada byte pertama dan kedua ke dalam eax. Dengan ini maka 4 byte dalam register eax telah berisi harga COL.

```
        mov     ecx,16000
        rep     stosd
```

Karena register eax telah memiliki 4 byte harga COL maka tinggal dilakukan proses transfer data 32 bit sejumlah 16000 kali ke virtual screen.

```
        mov     esp,ebp
        pop     ebp
        ret

ClearDouble     ENDP
```

Fungsi VESA_MoveDouble

Fungsi ini digunakan untuk memindahkan virtual screen ke virtual screen yang lain. Fungsi ini hanya digunakan untuk mode graphic selain 320 x 200 256 warna. Ukuran virtual screen yang akan dipindahkan secara otomatis akan diatur sesuai dengan mode graphic yang sedang dipilih yaitu dengan digunakannya variabel W_Scr yang berisi jumlah memory yang dibutuhkan pada mode graphic yang sedang dipakai dalam unit 32 bit.

Variabel ini akan otomatis diisi pada saat kita memanggil fungsi OpenVGA256.

```
VESA_MoveDouble PROC NEAR
        push    ebp
        mov     ebp,esp
        ARG     SOURCE:DWORD,DEST:DWORD
```

Parameter yang diterima ialah SOURCE yang berisi alamat virtual screen yang akan dipindahkan dan DEST yang berisi alamat virtual screen yang akan diisi.

```
        mov     esi,[SOURCE]
        mov     edi,[DEST]
```

Arahkan esi dan edi ke SOURCE dan DEST.

```
        mov     ecx,[W_Scr]
        rep     movsd
```

Lakukan transfer data 32 bit sejumlah W_Scr.

```
        mov     esp,ebp
        pop     ebp
        ret

VESA_MoveDouble ENDP
```

Fungsi VESA_ClearDouble

Fungsi ini digunakan untuk menghapus virtual screen dengan sebuah warna. Hanya digunakan untuk mode graphic selain 320 x 200 256 warna.

```
VESA_ClearDouble        PROC NEAR
        push    ebp
        mov     ebp,esp
        ARG     SOURCE:DWORD,COL:BYTE
```

Parameter yang diterima ialah SOURCE yang berisi alamat virtual screen yang akan dihapus, dan COL yang berisi warna yang akan digunakan untuk menghapus.

```
        mov     edi, [SOURCE]
        xor     eax,eax
        mov     ah,[COL]
        mov     al,[COL]
```

Isi register AX dengan COL pada kedua bytenya.

```
        mov     edx,eax
        shl     eax,16
        add     eax,edx
```

Isi 4 byte pada register EAX dengan COL seperti yang kita lakukan pada fungsi ClearDouble.

```
        mov     ecx,[W_Scr]
        rep     stosd
```

Lakukan transfer data 32 bit sejumlah W_Scr.

```
        mov     esp,ebp
        pop     ebp
```

```
        ret

VESA_ClearDouble        ENDP
```

Fungsi VESA_MoveToScreen

Fungsi ini digunakan untuk memindahkan sebuah virtual screen ke layar monitor. Hanya digunakan untuk mode graphic selain 320 x 200 256 warna.

```
VESA_MoveToScreen PROC NEAR
        push    ebp
        mov     ebp,esp
        ARG     SOURCE:DWORD
```

Parameter yang diterima ialah SOURCE yang berisi alamat virtual screen yang akan dipindahkan.

```
        mov     ecx,[W_Loop]
        mov     esi, [SOURCE]
```

Isikan ecx = W_Loop sebagai counter. Variabel W_Loop ini berisi jumlah jendela-1 yang dipergunakan oleh mode graphic yang sedang dipilih. Kemudian isikan esi dengan SOURCE.

```
      @rep:
        xor     ebx,ebx
        mov     edx,[W_Loop]
        sub     edx,ecx
```

Kurangi W_Loop dengan counter ecx sehingga didapat nomor jendela yang sedang diakses dan masukkan ke dalam register edx.

```
        mov     eax,00004f05h
```

Masukkan 4f05 ke dalam register eax untuk memilih servis interrupt DOS bagi penggantian jendela aktif dengan nomor jendela dalam register edx.

```
        int     10h
```

Panggil interrupt DOS. Jendela yang aktif sekarang adalah yang ditunjukkan pada register edx di atas.

```
        push    ecx
```

Simpan counter ecx dalam stack.

```
        mov     edi, 0a0000h
```

```
        mov     ecx, 16384
        rep     movsd
```

Lakukan transfer data dari SOURCE ke memory segment $A000 sejumlah 65536 Byte (16384 x 4 = 65536 Byte).

```
        pop     ecx
```

Kembalikan counter jendela ecx.

```
      loop    @rep
```

Loop hingga sampai pada jendela terakhir.

```
        mov     edi, 0a0000h
        mov     ecx, [W_Acc]
        cmp     ecx,0
        jz      @noaccess
```

Pada jendela terakhir, jika jumlah sisa byte = 0 maka proses selesai, jika jumlah sisa byte masih ada maka lakukan transfer data sejumlah W_Acc (variabel ini menunjukkan jumlah byte pada jendela terakhir).

```
        mov     eax,00004f05h
        xor     ebx,ebx
        mov     edx,[W_Loop]
        int     10h
```

Aktifkan jendela terakhir (nomor jendela terakhir = W_Loop).

```
        rep     movsd
```

Lakukan transfer data sejumlah ecx (ecx = W_Acc).

```
@noaccess:
        mov     esp,ebp
        pop     ebp
        ret

VESA_MoveToScreen ENDP
```

Fungsi VESA_ClearScreen

Fungsi ini digunakan untuk menghapus layar monitor secara langsung. Hanya digunakan pada mode graphic di atas 320 x 200 256 warna. Adapun proses pembuatannya hampir sama dengan fungsi VESA_MoveToScreen di atas, hanya perbedaannya ialah tidak dilakukannya pemindahan data dari virtual screen ke jendela-jendela

card VGA melainkan dilakukan pengisian warna ke jendela-jendela tersebut.

```
VESA_ClearScreen  PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     Col:BYTE
```

Parameter yang dikirimkan ialah COL yang berisi warna yang akan diisikan.

```
        mov     ecx,[W_Loop]
```

Lakukan counter loop untuk nomor jendela sejumlah W_Loop.

```
      @CS_rep:
        mov     eax,00004f05h
        mov     ebx,0
        mov     edx,[W_Loop]
        sub     edx,ecx
        int     10h
```

Aktifkan jendela yang akan diakses dengan pemanggilan interrupt.

```
        push    ecx
```

Simpan ecx.

```
        mov     edi, 0a0000h
        mov     ecx, 16384
```

Set counter sejumlah 65536 byte (16384 x 4 = 65536 byte).

```
        xor     eax,eax
        mov     al,[Col]
        mov     ah,[Col]
        mov     edx,eax
        shl     eax,16
        add     eax,edx
```

Isi 4 byte pada register eax dengan Col.

```
        rep     stosd
```

Lakukan transfer data untuk mengisi jendela yang sedang aktif dengan warna Col.

```
        pop     ecx
```

Kembalikan harga register ecx.

```
loop    @CS_rep

        mov     edi, 0a0000h
        mov     ecx, [W_Acc]

        mov     eax,00004f05h
        mov     ebx,0
        mov     edx,[W_Loop]
        int     10h
```

Aktifkan jendela nomor terakhir.

```
        xor     eax,eax
        mov     ah,[Col]
        mov     al,[Col]
        mov     edx,eax
        shl     eax,16
        add     eax,edx
        rep     stosd
```

Isi jendela terakhir dengan Col.

```
        mov     esp,ebp
        pop     ebp
        ret

VESA_ClearScreen ENDP
```

Fungsi OpenVGA256

Fungsi ini digunakan untuk membuka mode graphic 8 bit dengan mode resolusi yang dapat dipilih yaitu:

- 320 x 200 256 warna
- 640 x 400 256 warna
- 640 x 480 256 warna
- 800 x 600 256 warna
- 1024 x 768 256 warna

```
OpenVGA256 PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     mode:DWORD
```

Parameter yang dikirimkan ialah mode graphic yang dipilih. Untuk itu Anda dapat melihat Tabel 5.1.

Tabel 5.1 Mode Graphic VGA.ASM

Mode Graphic	Mode yang dikirimkan
320 x 200 256 warna	$13 atau 19 desimal
640 x 400 256 warna	256 desimal
640 x 480 256 warna	257 desimal
800 x 600 256 warna	259 desimal
1024 x 768 256 warna	261 desimal

```
        cmp     [mode],13h
        je      @mode13
        mov     eax,4f02h
        mov     ebx,[mode]
        int     10h
        mov     ebx,[mode]
        cmp     ebx,256
        je      @mode256
        cmp     ebx,259
        je      @mode259
        cmp     ebx,257
        je      @mode257
        cmp     ebx,261
        je      @mode261
        jmp     @finmode
@mode13:
        mov     eax,13h
        int     10h
        mov     [S_L],319
        mov     [S_W],199
        mov     [A_SL],320
        mov     [A_SW],200
        mov     [W_Scr],16000
        mov     [W_Loop],0
        mov     [W_Acc],0
        jmp     @finmode
```

Untuk mode graphic 320 x 200 256 warna:

```
S_L     = Koordinat Xmax = 319
S_W     = Koordinat Ymax = 199
```

```
A_SL   = Jumlah titik horizontal = 320
A_SW   = Jumlah titik vertikal   = 200
W_Scr  = Jumlah memory yang dipergunakan = 16000 x 4
W_Loop = Nomor jendela terakhir = 1
W_Acc  = Jumlah byte pada jendela terakhir = 0
```

Pada mode graphic ini, variabel-variabel di atas sebenarnya tidak akan dipergunakan karena beberapa fungsi yang membutuhkan variabel di atas seperti MoveDouble, ClearDouble dan MoveToScreen telah dirakit secara terpisah.

```
@mode256:
  mov     [S_L],639
  mov     [S_W],399
  mov     [A_SL],640
  mov     [A_SW],400
  mov     [W_Scr],64000
  mov     [W_Loop],3
  mov     [W_Acc],14848
  jmp     @finmode
```

Untuk mode graphic 640 x 400 256 warna:

```
S_L    = Koordinat Xmax = 639
S_W    = Koordinat Ymax = 399
A_SL   = Jumlah titik horizontal = 640
A_SW   = Jumlah titik vertikal   = 400
W_Scr  = Jumlah memory yang dipergunakan = 64000 x 4
W_Loop = Nomor jendela terakhir = 3
W_Acc  = Jumlah byte pada jendela terakhir = 14848
```

```
@mode257:
  mov     [S_L],639
  mov     [S_W],479
  mov     [A_SL],640
  mov     [A_SW],480
  mov     [W_Scr],76800
  mov     [W_Loop],4
  mov     [W_Acc],11264
  jmp     @finmode
```

Untuk mode graphic 640 x 480 256 warna:

```
S_L    = Koordinat Xmax = 639
S_W    = Koordinat Ymax = 479
```

```
A_SL   = Jumlah titik horizontal = 640
A_SW   = Jumlah titik vertikal   = 480
W_Scr  = Jumlah memory yang dipergunakan = 76800 x 4
W_Loop = Nomor jendela terakhir = 4
W_Acc  = Jumlah byte pada jendela terakhir = 11264
```

```
@mode259:
  mov     [S_L],799
  mov     [S_W],599
  mov     [A_SL],800
  mov     [A_SW],600
  mov     [W_Scr],120000
  mov     [W_Loop],7
  mov     [W_Acc],5312
  jmp     @finmode
```

Untuk mode graphic 800 x 600 256 warna:

```
S_L    = Koordinat Xmax = 799
S_W    = Koordinat Ymax = 599
A_SL   = Jumlah titik horizontal = 800
A_SW   = Jumlah titik vertikal   = 600
W_Scr  = Jumlah memory yang dipergunakan = 120000 x 4
W_Loop = Nomor jendela terakhir = 7
W_Acc  = Jumlah byte pada jendela terakhir = 5312
```

```
@mode261:
  mov     [S_L],1023
  mov     [S_W],767
  mov     [A_SL],1024
  mov     [A_SW],768
  mov     [W_Scr],196608
  mov     [W_Loop],12
  mov     [W_Acc],0
```

Untuk mode graphic 1024 x 768 256 warna:

```
S_L    = Koordinat Xmax = 1023
S_W    = Koordinat Ymax = 767
A_SL   = Jumlah titik horizontal = 1024
A_SW   = Jumlah titik vertikal   = 768
W_Scr  = Jumlah memory yang dipergunakan = 196608 x 4
W_Loop = Nomor jendela terakhir = 12
W_Acc  = Jumlah byte pada jendela terakhir = 0
```

```
@finmode:
  mov     [V_X1],0
  mov     [V_Y1],0
  mov     eax,[S_L]
  mov     [V_X2],eax
  mov     eax,[S_W]
  mov     [V_Y2],eax
```

Isikan koordinat ViewPort = (0,0)-(S_L, S_W) sebagai default.

```
  mov     esp,ebp
  pop     ebp
  ret

OpenVGA256 ENDP
```

Fungsi CloseVGA256

Fungsi ini digunakan untuk menutup mode graphic 8 bit.

```
CloseVGA256     PROC NEAR

  mov     ax,2
  int     10h
```

Gunakan interrupt DOS untuk kembali ke mode text.

```
  ret

CloseVGA256     ENDP
```

Nah, sekarang marilah kita mencoba fungsi-fungsi yang telah kita bahas di atas ke dalam program.

Coba jalankan program 2SCR1.EXE dalam directory EXE.

Pada awal program, akan dibuka mode graphic 640 x 400 256 warna dan ditampilkan layar berwarna putih. Kemudian tekan Enter, layar akan berganti warna menjadi biru. Penekanan Enter sekali lagi menyebabkan program berakhir.

Sebenarnya program ini hanya untuk mencoba fungsi animasi dua layar yang dimiliki oleh library VGA.ASM. Pada awal program, diciptakan virtual screen yang diberi warna biru. Sedangkan layar tampilan diisi dengan warna putih. Setelah tombol Enter ditekan, virtual screen akan dipindahkan ke layar tampilan dengan fungsi VESA_MoveToScreen.

Berikut ini ialah cara meng-compile unit VGA.ASM beserta program 2SCR1.CPP menjadi executable file (untuk pentium). Jika para pembaca menggunakan komputer jenis lain dari pentium, ganti parameter /fp5 dan /5s sesuai dengan jenis komputer yang digunakan, seperti yang telah dibahas pada penggunaan utility watcom C.

```
wpp386 /d3 /5s /fp5 /zt 2SCR1.CPP
tasm /mx /zi /os vga.asm vga.obj
wlink system dos4g debug all file 2SCR1,vga name 2SCR1
```

Ingat, tasm harus berada dalam satu directory dengan file 2SCR1.CPP dan VGA.ASM yang akan di-compile.

PENJELASAN PROGRAM

Listing program ini dapat ditemukan dalam directory SOURCE.

```
/* ------------------------------------------------------------
Nama File :  2SCR1.CPP
Definisi  :  Program mencoba fungsi-fungsi standar VGA.ASM
             pada mode graphic 640 x 400 256 warna
Programmer:  Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
```

Definisikan penggunaan header file try.h sebagai interface ke unit VGA.ASM.

```
char ch;
char * Buffer;
```

Deklarasikan variabel pointer Buffer sebagai virtual screen.

```
int main()
{
  Buffer=(char *)malloc(MEM400);
```

Pesan memory sebesar MEM400 (di dalam file header TRY.H, MEM400 ialah konstanta berharga 256000 yang menunjukkan jumlah memory yang diperlukan oleh virtual screen pada mode graphic 640 x 400 256 warna). Untuk mode-mode graphic yang lain, gunakan Tabel 5.2.

```
OpenVGA256(MOD400);
```

Buka mode graphic 640 x 400 256 warna dengan memanggil fungsi OpenVGA256 dengan parameter MOD400, yang dalam file header TRY.H, MOD400 ialah konstanta berharga 256.

Untuk mode-mode graphic yang lain, gunakan Tabel 5.2.

Tabel 5.2 Konstanta Mode Graphic dan Memory

Mode graphic	Konstanta untuk memory	Konstanta mode graphic
320 x 200 256 warna	MEM320	MOD320
640 x 400 256 warna	MEM400	MOD400
640 x 480 256 warna	MEM640	MOD640
800 x 600 256 warna	MEM800	MOD800
1024 x 768 256 warna	MEM1024	MOD1024

```
VESA_ClearScreen(15);
```

Hapus layar tampilan dengan warna 15 = putih.

```
ch=getch();
```

Tunggu penekanan tombol keyboard.

```
VESA_ClearDouble(Buffer,1);
```

Hapus virtual screen Buffer dengan warna 1 = biru.

```
VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen Buffer ke layar monitor.

```
ch=getch();
```

Tunggu penekanan tombol keyboard.

```
CloseVGA256();
return 1;
}
```

Program di atas menggunakan mode graphic 640 x 400 256 warna, sekarang kita lihat perbedaannya jika kita hendak menggunakan mode graphic 320 x 200 256 warna. Jalankan program 2SCR2.EXE.

Jalannya program sama persis seperti program 2SCR1.EXE. Hanya bedanya, sekarang digunakan mode graphic 320 x 200 256 warna. Bagaimana membedakannya dari tampilan di layar monitor? Sulit, bahkan hampir tidak mungkin, karena layar hanya kita isi dengan warna polos sehingga kita tidak dapat mengetahui ukuran sebuah titik pada layar monitor. Tentu saja titik yang lebih kecil berarti digunakannya resolusi yang lebih tinggi. Semestinya lebih baik jika kita tampilkan sebuah titik pada layar monitor, tetapi kita belum sampai pada pembahasan fungsi PutPixel, jadi yah untuk sekarang, percaya sajalah bahwa itu adalah resolusi 320 x 200 256 warna, pengarang tidak bohong kok.

Kalau di tampilan tidak berbeda, belum tentu di programnya sama. Coba ikuti pembahasan berikut.

PENJELASAN PROGRAM

```
/* ------------------------------------------------------
Nama File : 2SCR2.CPP
Definisi  : Program mencoba fungsi-fungsi standar VGA.ASM
            pada mode graphic 320 x 200 256 warna
Programmer: Agustinus Nalwan
------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"

char ch;
char * Buffer;
```

Deklarasikan variabel pointer untuk virtual screen.

```
int main()
{
  Buffer=(char *)malloc(MEM320);
```

Virtual screen diciptakan dengan memesan memory seukuran MEM320 (lihat Tabel 5.2).

```
  OpenVGA256(MOD320);
```

Mode graphic 320 x 200 256 warna dibuka dengan mengirimkan parameter MOD320 (lihat Tabel 5.2).

```
  ClearDouble(SCREEN,15);
```

Hapus layar tampilan dengan langsung mengirimkan alamat memory segment $A000 yang berada pada konstanta SCREEN (lihat file TRY.H) sebagai virtual screen yang akan dihapus. Di sini digunakan warna nomor 15 = putih.

Kalau pada program terdahulu dengan mode graphic 640 x 400 256 warna, kita menggunakan fungsi VESA_ClearDouble, di sini di mode 320 x 200 256 warna kita harus menggunakan fungsi ClearDouble.

```
  ch=getch();
```

Tunggu tombol keyboard ditekan.

```
  ClearDouble(Buffer,1);
```

Hapus virtual screen dengan warna 1 = biru.

```
  MoveToScreen(Buffer);
```

Tampilkan virtual screen ke layar monitor. Fungsi yang digunakan lain dengan yang digunakan pada mode graphic 640 x 400 256 warna.

Dulu digunakan VESA_MoveToScreen, sekarang kita menggunakan MoveToScreen.

```
  ch=getch();
```

Tunggu tombol keyboard ditekan.

```
  CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

5.4 Fungsi-fungsi Penampilan Bangun Geometri

Selanjutnya akan dibahas tentang fungsi-fungsi penampilan bangun geometri seperti penampilan titik, balok, garis dan lain sebagainya. Fungsi-fungsi ini kebanyakan telah kita miliki dalam unit VGA-CAD.TPU pada seri Pemrograman Animasi dan Game Profesional sebelumnya.

Fungsi VESA_PutPixel

Fungsi ini digunakan untuk meletakkan sebuah titik pada virtual screen.

```
VESA_PutPixel          PROC NEAR

        push   ebp
        mov    ebp,esp
        ARG    BUFFER:DWORD, X:DWORD, Y:DWORD, COL:BYTE
```

Parameter yang dikirimkan ialah BUFFER yaitu alamat virtual screen tempat titik akan diletakkan. X dan Y yaitu koordinat X dan Y tempat titik akan diletakkan, dan COL yaitu warna dari titik yang akan diletakkan.

```
        mov      eax,[Y]
        mul      [A_SL]
        add      eax,[X]
```

Hitung jumlah byte antara kiri atas virtual screen ke titik yang akan diletakkan. Jumlah byte ini selanjutnya akan dinamakan offset titik.

Adapun cara perhitungannya ialah dengan rumus:

Offset Titik = Y * A_SL + X

```
        mov      edi,[BUFFER]
```

Arahkan edi ke virtual screen di mana titik akan diletakkan.

```
        mov      cl,COL
        mov      BYTE PTR [edi+eax],cl
```

Isi memory lokasi edi (alamat virtual screen) + eax (offset titik) dengan warna titik yang disimpan dalam register cl.

```
        mov      esp,ebp
        pop      ebp

        ret

VESA_PutPixel          ENDP
```

Nah, sekarang kita akan mencoba sebuah program yang akan memanfaatkan fungsi VESA_PutPixel ini. Program ini ialah semacam program sederhana untuk menggambar pada layar monitor yang mendukung mode resolusi 320 x 200, 640 x 400, 640 x 480, 800 x 600 dan 1024 x 768 256 warna.

Coba jalankan program DRAW.EXE.

Pada saat awal program ini dijalankan kita diharapkan untuk memilih mode resolusi dengan menekan angka 1-5. Coba pilih salah satu mode resolusi yang didukung oleh card VGA Anda dan tekan ENTER. Selanjutnya akan terlihat pilihan warna pada layar dan juga terdapat sebuah kotak putih yang berkedip yang mengelilingi salah satu warna. Warna inilah yang sedang aktif. Untuk mengganti warna tekan tombol spasi. Untuk menggambar gerakkan kursor ke atas, bawah, kiri dan kanan. Untuk keluar tekan ESC.

PENJELASAN PROGRAM

```
/*     ------------------------------------------------------
Nama File : DRAW.CPP
Definisi  : Program penyuntingan gambar sederhana
Programmer: Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"

int main()
{
  int finish=0;
  int x=160,y=100,a,b,MaxX,MaxY;
```

Koordinat awal pena ialah (160,100).

```
  char color=15;
```

Warna awal ialah nomor 15 = putih.

```
  char ctr=0;
  char ch,blink;
  char * Buffer;
  int pilihan;

  printf("\n\n\n");
  printf("          1. 320  x 200 256 warna \n");
  printf("          2. 640  x 400 256 warna \n");
  printf("          3. 640  x 480 256 warna \n");
  printf("          4. 800  x 600 256 warna \n");
  printf("          5. 1024 x 768 256 warna \n");
```

```
printf("        Pilihan: ? ");
scanf("%d",&pilihan);
```

Sediakan menu untuk memilih mode graphic.

```
if (pilihan==1)
{
  OpenVGA256(MOD320);
  Buffer=(char *)malloc(MEM320);
  MaxX=320;MaxY=200;
```

Mode graphic 320 x 200 256 warna dipilih. Ciptakan virtual screen dan isikan MaxX dan MaxY sebagai jumlah titik horizontal dan vertikal.

```
} else
if (pilihan==2)
{
  OpenVGA256(MOD400);
  Buffer=(char *)malloc(MEM400);
  MaxX=640;MaxY=400;
```

Mode graphic 640 x 400 256 warna dipilih.

```
} else
if (pilihan==3)
{
  OpenVGA256(MOD640);
  Buffer=(char *)malloc(MEM640);
  MaxX=640;MaxY=480;
```

Mode graphic 640 x 480 256 warna dipilih.

```
} else
if (pilihan==4)
{
  OpenVGA256(MOD800);
  Buffer=(char *)malloc(MEM800);
  MaxX=800;MaxY=600;
```

Mode graphic 800 x 600 256 warna dipilih.

```
} else
if (pilihan==5)
{
  OpenVGA256(MOD1024);
  Buffer=(char *)malloc(MEM1024);
  MaxX=1024;MaxY=768;
```

Mode graphic 1024 x 768 256 warna dipilih.

```
} else
return 0;

VESA_ClearDouble(Buffer,0);
```

Hapus virtual screen dengan warna hitam.

```
while (!finish)
{
  for (a=0;a<64;a++)
  {
    for (b=0;b<4;b++)
    {
      VESA_Block(Buffer,a*4,b*3+MaxY-20,a*4+3,b*3+MaxY-
              18,ctr);
      if (color==ctr)
      {
      VESA_Rect(Buffer,a*4,b*3+MaxY-20,a*4+3,b*3+MaxY-
              18,blink);
      }
      ctr++;
    }
  }
```

Tampilkan pilihan warna dan kotak pointer yang mengelilingi warna yang sedang dipilih.

```
blink++;
VESA_PutPixel(Buffer,x,y,color);
```

Tampilkan titik di layar pada koordinat pena (x,y).

```
if (pilihan==1)
  MoveToScreen(Buffer);
```

Jika mode graphic yang dipakai adalah 320 x 200 256 warna maka pakailah fungsi MoveToScreen untuk memindahkan virtual screen ke layar.

```
else
  VESA_MoveToScreen(Buffer);
```

Untuk mode graphic yang lain, pergunakan VESA_MoveToScreen.

```
while (!kbhit())
{
}
```

```
    ch=getch();
    if (ch==72) y--;
    if (ch==80) y++;
    if (ch==75) x--;
    if (ch==77) x++;
```

Jika kursor ditekan, gerakkan pena sesuai tombol yang ditekan.

```
    if (ch==' ') color++;
    if (ch==27) finish=1;
```

Jika spasi ditekan, ubah warna yang digunakan dan jika ESC ditekan, keluar dari program.

```
  }
  CloseVGA256();

Tutup mode graphic.
  return 1;
}
```

Fungsi Block

Fungsi ini digunakan untuk meletakkan balok pada virtual screen.

```
VESA_Block              PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,
                X2:DWORD,Y2:DWORD,COL:BYTE
```

Parameter yang dikirimkan ialah BUFFER yaitu alamat memory virtual screen di mana balok akan diletakkan. X1, Y1, X2, Y2 adalah koordinat balok yang akan diletakkan, dan COL yaitu warna balok yang akan diletakkan.

Pada proses selanjutnya akan dilakukan pendeteksian terhadap batas kiri, kanan, atas dan bawah dari virtual screen. Jika balok berada di luar virtual screen, balok tidak akan digambar ke layar.

```
        mov     eax,[X1]
        cmp     eax,[S_L]
        jg      @BL_Vin
```

Deteksi, apakah seluruh bagian balok menerobos batas kanan layar (dengan cara melihat apakah koordinat X1 yang merupakan bagian paling kiri balok lebih besar dari S_L yang merupakan koordinat hori-

zontal maksimum). Jika ya, batalkan proses penggambaran balok karena balok berada di luar layar.

```
mov     eax,[X2]
cmp     eax,0
jl      @BL_Vin
```

Deteksi, apakah seluruh bagian balok menerobos batas kiri layar (dengan cara melihat apakah koordinat X2 yang merupakan bagian paling kanan balok lebih kecil dari 0 yang merupakan koordinat horizontal minimum). Jika ya, batalkan proses penggambaran balok karena balok berada di luar layar.

```
mov     eax,[Y1]
cmp     eax,[S_W]
jg      @BL_Vin
```

Deteksi, apakah seluruh bagian balok menerobos batas atas layar (dengan cara melihat apakah koordinat Y1 yang merupakan bagian paling atas balok lebih besar dari S_W yang merupakan koordinat vertikal maksimum). Jika ya, batalkan proses penggambaran balok karena balok berada di luar layar.

```
mov     eax,[Y2]
cmp     eax,0
jl      @BL_Vin
```

Deteksi, apakah seluruh bagian balok menerobos batas bawah layar (dengan cara melihat apakah koordinat Y2 yang merupakan bagian paling bawah balok lebih kecil dari 0 yang merupakan koordinat vertikal minimum). Jika ya, batalkan proses penggambaran balok karena balok berada di luar layar.

Proses selanjutnya ialah memotong bagian-bagian balok yang berada di luar virtual screen.

```
mov     eax,[X1]
cmp     eax,0
jge     @VESx1_not_negs
```

Jika balok tidak terpotong oleh batas kiri layar, lompati proses di bawah ini.

```
mov     [x1],0
```

Potong balok sehingga koordinat X1 yang baru menjadi koordinat X = 0.

```
@VESx1_not_negs:
  mov     eax,[Y1]
  cmp     eax,0
  jge     @VESy1_not_negs
```

Jika balok tidak terpotong oleh batas atas layar, lompati proses di bawah ini.

```
  mov     [y1],0
```

Potong balok sehingga koordinat Y1 dari balok menjadi Y = 0.

```
@VESy1_not_negs:
  mov     eax,[X2]
  cmp     eax,[S_L]
  jle     @VESx2_not_pass
```

Jika balok tidak terpotong oleh batas kanan layar, lompati proses di bawah ini.

```
  mov     eax,[S_L]
  mov     [X2],eax
```

Potong balok sehingga koordinat X2 menjadi X = S_L.

```
@VESx2_not_pass:
  mov     eax,[Y2]
  cmp     eax,[S_W]
  jle     @VESy2_not_pass
```

Jika balok tidak terpotong oleh batas bawah layar, lompati proses di bawah ini.

```
  mov     eax,[S_W]
  mov     [y2],eax
```

Potong balok sehingga koordinat Y2 menjadi Y = S_W.

Pada proses selanjutnya akan dilakukan pengisian titik pada balok, dengan asumsi koordinat balok telah dibatasi dengan benar setelah melewati proses yang panjang di atas.

```
@VESy2_not_pass:
  mov     eax,[x2]
  sub     eax,[x1]
  inc     eax
  mov     edx,eax
```

Hitung panjang balok.

```
and     eax,000000003h
```

Hitung panjang balok mod 4. Dan masukkan hasilnya dalam eax.

Perhitungan ini dilakukan karena nantinya akan dilakukan transfer data 32 bit (4 byte) sehingga jika panjang balok bukan kelipatan 4 maka akan ada sisa beberapa byte yang tidak terisi. Sisa byte ini ditunjukkan dalam register eax dengan hasil perhitungan di atas.

```
shr     edx,2
```

Bagi panjang balok dengan 4 sehingga diperoleh panjang balok dalam satuan 32 bit.

```
push    eax
```

Simpan panjang balok mod 4 ke dalam stack.

```
mov     ebx,[y2]
sub     ebx,[y1]
inc     ebx
```

Hitung lebar balok.

```
Push    edx
```

Masukkan edx ke dalam stack.

```
mov     eax,[y1]
mul     [A_SL]
add     eax,[x1]
```

Hitung offset (offset di sini sama artinya dengan offset pada PutPixel, hanya saja offset di sini menunjuk ke titik kiri atas/pertama dari balok) balok.

```
pop     edx
mov     edi,[BUFFER]
add     edi,eax
```

Tambahkan alamat virtual screen dengan offset dari balok.

```
xor     eax,eax
mov     al,[col]
mov     ah,al
mov     ecx,eax

shl     eax,16
add     eax,ecx
```

Isikan 4 byte pada register eax dengan Col.

```
        mov     ecx,ebx
```

Isikan ecx dengan lebar balok sebagai counter vertikal.

```
        cld
@VES1s:
        mov     ebx,ecx
```

Kopikan counter vertikal ke dalam register ebx.

```
        mov     esi,edi
```

Kopikan harga register edi ke esi sehingga kita tidak kehilangan harga register edi sebelum dilakukannya proses rep stosd.

```
        mov     ecx,edx
```

Isikan panjang balok (dalam satuan 32 bit) ke dalam ecx sebagai counter horizontal.

```
        rep     stosd
```

Transfer data 32 bit dari eax ke dalam virtual screen sejumlah panjang balok.

```
        pop     ecx
        push    ecx
```

Ambil sisa byte tadi yang disimpan dalam stack dan masukkan lagi ke dalam stack. Tujuannya ialah agar harga tersebut disimpan dalam register ecx tetapi masih tetap disimpan pula dalam stack.

```
        rep     stosb
```

Lakukan transfer data byte untuk mengisi sisa byte horizontal yang masih kosong.

```
        mov     edi,esi
```

Kembalikan harga register edi seperti sebelum melakukan rep stosd dan rep stosb di atas.

```
        add     edi,[A_SL]
```

Tambahkan register edi dengan A_SL untuk mengakses koordinat Y berikutnya.

```
        mov     ecx,ebx
```

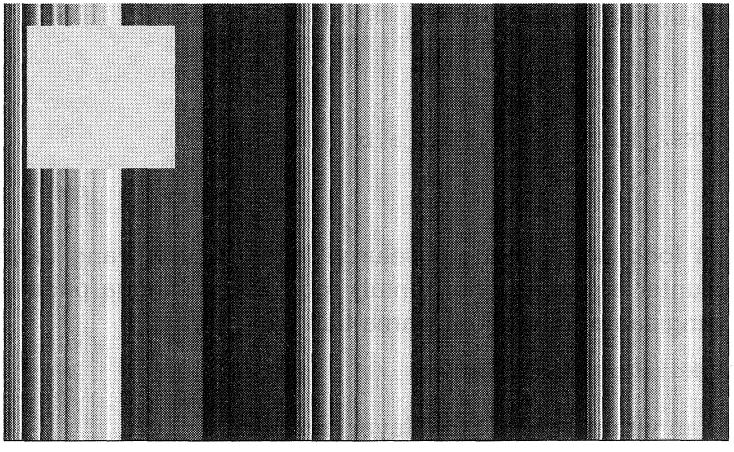
Kopikan lebar balok ke dalam ecx sebagai counter vertikal.

```
        loop @VES1s
@BL_Vin:
          mov       esp,ebp
          pop       ebp

          ret

VESA_Block              ENDP
```

Nah, sekarang kita akan mencoba fungsi-fungsi yang telah kita pelajari di atas. Coba jalankan program BIDANG1.EXE.

Sekarang baru kita percaya bahwa, mode graphic 640 x 400 256 warna telah ditampilkan dengan benar. Gambar 5.2 menunjukkan output program ini.

Seluruh layar akan diisi dengan titik-titik mulai warna 0 hingga 255 dengan menggunakan fungsi VESA_PutPixel. Coba tekan ENTER, akan ditampilkan sebuah balok yang berwarna coklat muda dengan menggunakan fungsi VESA_Block.



Gambar 5.2 Output Program BIDANG1.EXE.

PENJELASAN PROGRAM

```
/*      ------------------------------------------------------
Nama File : BIDANG1.CPP
Definisi  : Program mencoba bangun-bangun geometri
Programmer: Agustinus Nalwan
------------------------------------------------------------ */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
```

Definisikan header file TRY.H.

```
int main()
{
  char ch;
  char * Buffer;
  int a,b;
  OpenVGA256(MOD400);
```

Buka mode graphic 640 x 400 256 warna.

```
  Buffer=(char *)malloc(MEM400);
```

Ciptakan virtual screen pada variabel Buffer.

```
  VESA_ClearDouble(Buffer,0);
```

Hapus virtual screen dengan warna 0 = hitam.

```
  for (a=0;a<639;a++)
  {
    for (b=0;b<399;b++)
    {
      VESA_PutPixel(Buffer,a,b,a%256);
    }
  }
```

Lakukan loop untuk mengisi seluruh layar. Warna yang digunakan ialah counter a (counter horizontal) mod 256 sehingga pada koordinat X yang sama, dihasilkan warna titik yang sama.

```
  VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen pada layar monitor.

```
  ch=getch();
```

Tunggu penekanan tombol keyboard.

```
  VESA_Block(Buffer,20,20,150,150,90);
```

Letakkan balok dengan koordinat (20,20)-(150,150) dengan warna 90 pada virtual screen.

```
  VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen pada layar monitor.

```
ch=getch();
```

Tunggu penekanan tombol keyboard.

```
CloseVGA256();
```

Tutup mode graphic.

```
return 1;
}
```

Fungsi VESA_VWP_Block

Fungsi ini sama persis seperti VESA_Block hanya saja di sini dipertimbangkan ukuran dari viewport, sehingga balok akan dipotong terhadap batas-batas viewport bukannya batas-batas layar.

```
VESA_VWP_Block          PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,
                X2:DWORD,Y2:DWORD,COL:BYTE
        mov     eax,[X1]
        cmp     eax,[V_X2]
        jg      @BL_V2in
        mov     eax,[X2]
        cmp     eax,[V_X1]
        jl      @BL_V2in
        mov     eax,[Y1]
        cmp     eax,[V_Y2]
        jg      @BL_V2in
        mov     eax,[Y2]
        cmp     eax,[V_Y1]
        jl      @BL_V2in
```

Perhatikan, pembatasan dilakukan terhadap variabel V_X1, V_X2, V_Y1, V_Y2 tidak seperti pada fungsi VESA_Block yang menggunakan pembatasan koordinat minimum dan maksimum dari layar.

```
        mov     eax,[X1]
        cmp     eax,[V_X1]
        jge     @VESx12_not_negs
        mov     eax,[V_X1]
        mov     [x1],eax
```

```
@VESx12_not_negs:
  mov     eax,[Y1]
  cmp     eax,[V_Y1]
  jge     @VESy12_not_negs
  mov     eax,[V_Y1]
  mov     [y1],eax
@VESy12_not_negs:
  mov     eax,[X2]
  cmp     eax,[V_X2]
  jle     @VESx22_not_pass
  mov     eax,[V_X2]
  mov     [X2],eax
@VESx22_not_pass:
  mov     eax,[Y2]
  cmp     eax,[V_Y2]
  jle     @VESy22_not_pass
  mov     eax,[V_Y2]
  mov     [y2],eax
```

Juga pemotongan balok yang menerobos batas layar dilakukan dengan menggunakan variabel-variabel viewport.

```
@VESy22_not_pass:
  mov     eax,[x2]
  sub     eax,[x1]
  inc     eax
  mov     edx,eax
  and     eax,000000003h
  shr     edx,2
  push    eax

  mov     ebx,[y2]
  sub     ebx,[y1]
  inc     ebx

  mov     eax,[y1]
  push    edx
  mul     [A_SL]
  add     eax,[x1]
  pop     edx
  mov     edi,[BUFFER]
  add     edi,eax

  xor     eax,eax
  mov     al,[col]
  mov     ah,al
  mov     ecx,eax
```

```
        shl     eax,16
        add     eax,ecx

        mov     ecx,ebx
        cld
      @VES12s:
        mov     ebx,ecx
        mov     esi,edi
        mov     ecx,edx
        rep     stosd
        pop     ecx
        push    ecx
        rep     stosb
        mov     edi,esi
        add     edi,[A_SL]
        mov     ecx,ebx
      loop @VES12s
@BL_V2in:
        mov     esp,ebp
        pop     ebp
        ret

VESA_VWP_Block          ENDP
```

Proses-proses lainnya adalah sama persis dengan fungsi VESA_Block.

Gunakan program BIDANG1B.EXE untuk mencoba fungsi di atas.

Akan ditampilkan sebuah persegi panjang putih dan sebuah balok yang memotong persegi panjang tersebut. Ukuran viewport diatur sama persis seperti persegi panjang putih tersebut. Tetapi karena digunakannya fungsi VESA_Block yang tidak mempedulikan viewport, balok tidak dipotong terhadap viewport. Sekarang coba tekan ENTER. Layar akan dihapus dan kembali ditampilkan persegi panjang putih. Kemudian balok akan ditampilkan dengan menggunakan fungsi VESA_VWP_Block pada koordinat yang sama. Sekarang terlihat bahwa balok terpotong oleh viewport dengan benar.

PENJELASAN PROGRAM

```
/* ------------------------------------------------------
Nama File : BIDANG1B.CPP
Definisi  : Program mencoba fungsi Block dengan viewport
Programmer: Agustinus Nalwan
------------------------------------------------------ */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"

int main()
{
  char ch;
  char * Buffer;
  int a,b;
  OpenVGA256(MOD400);
  Buffer=(char *)malloc(MEM400);
  VESA_ClearDouble(Buffer,0);
```

Hapus virtual screen.

```
  VESA_Rect(Buffer,50,50,250,250,14);
```

Tampilkan persegi panjang sebagai tanda ukuran viewport.

```
  ViewPort(50,50,250,250);
```

Ciptakan viewport yang seukuran dengan persegi panjang di atas.

```
  VESA_Block(Buffer,20,20,150,150,15);
  VESA_MoveToScreen(Buffer);
```

Tampilkan balok yang akan memotong viewport dan tampilkan virtual screen ke layar monitor.

```
  ch=getch();
```

Tunggu penekanan tombol keyboard.

```
  VESA_ClearDouble(Buffer,0);
```

Hapus virtual screen dengan warna 0 = hitam.

```
  VESA_Rect(Buffer,50,50,250,250,14);
```

Tampilkan persegi panjang.

```
  VESA_VWP_Block(Buffer,20,20,150,150,15);
```

Tampilkan balok menggnakan fungsi VESA_VWP_Block sehingga balok ini akan terpotong oleh viewport.

```
  VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen ke layar.

```
  ch=getch();
  CloseVGA256();
```

Tunggu penekanan keyboard dan tutup mode graphic.

```
  return 1;
}
```

Selanjutnya, masih dalam lingkup fungsi-fungsi penampilan bangun geometri, kita akan membahas fungsi penggambaran garis horizontal, vertikal dan garis bebas (multi direction).

Penggambaran garis lurus dengan garis bebas dibedakan. Garis lurus ialah garis horizontal atau garis vertikal (kedua titik penghubung garis ini memiliki salah satu koordinat Y atau X yang sama). Pada garis bebas, hal tersebut tidak selalu dipenuhi. Oleh karena itu, fungsi garis bebas cenderung lebih sulit dibuat dan lebih lambat kerjanya.

Fungsi VESA_Hline

Fungsi ini berguna untuk menggambar sebuah garis lurus horizontal.

```
VESA_HLine              PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,
                 X2:DWORD,COL:BYTE
```

Parameter yang dikirimkan ialah BUFFER yaitu alamat virtual screen di mana garis akan ditampilkan. X1,Y1 yaitu koordinat titik pertama garis, dan X2 yaitu koordinat X titik kedua. Kemudian COL yaitu warna garis yang akan digambar.

Sehingga jika kita ingin menggambar sebuah garis dengan koordinat (100, 100)-(180, 100) maka kita cukup memanggil VESA_Hline dengan (X1,Y1) = (100, 100) dan X2 = 180 karena Y2 sama dengan Y1 yaitu 100.

```
        mov     eax,[X1]
        cmp     eax,[X2]
        jle     @AANSwap
```

Periksa, apakah X1 lebih besar dari X2. Jika tidak, maka tidak perlu dilakukan swap antara X1 dan X2. Tujuannya ialah agar X2 selalu lebih besar dari X1 karena nantinya kita akan melakukan pengisian

titik sepanjang garis mulai dari koordinat X1 ke arah kanan.

```
mov     ebx,[X2]
mov     [X1],ebx
mov     [X2],eax
```

Lakukan proses swap antara X1 dan X2.

```
@AANSwap:
  mov     eax,[X1]
  cmp     eax,0
  jge     @HL_NoAddX
```

Deteksi, apakah garis berpotongan dengan batas kiri layar.

```
  mov     [X1],0
```

Jika ya, potong garis terhadap koordinat X = 0.

```
@HL_NoAddX:
  mov     eax,[X2]
  cmp     eax,S_L
  jle     @HL_NoAddX2
```

Deteksi, apakah garis berpotongan dengan batas kanan layar.

```
  mov     eax,[S_L]
  mov     [X2],eax
```

Jika ya, potong garis terhadap koordinat X = S_L (koordinat horizontal maksimum).

```
@HL_NoAddX2:
  mov     eax,[Y1]
  cmp     eax,0
```

Deteksi, apakah garis berada di atas layar.

```
  jl      @HL_Vin
```

Jika ya, batalkan penggambaran garis karena garis berada di luar layar.

```
  cmp     eax,S_W
```

Deteksi, apakah garis berada di bawah layar.

```
  jg      @HL_Vin
```

Jika ya, batalkan penggambaran garis karena garis berada di luar layar.

```
        mov     eax,[X2]
        sub     eax,[X1]
```

Hitung panjang antara X2 dan X1.

```
        cmp     eax,0
        jl      @HL_Vin
```

Jika panjang lebih kecil dari 0 maka batalkan penggambaran garis. Hal ini untuk meyakinkan bahwa koordinat X2 lebih besar dari X1.

```
        inc     eax
        mov     ecx,eax
```

Masukkan panjang garis ke dalam ecx sebagai counter.

```
        mov     eax,[Y1]
        mul     [A_SL]
        add     eax,[X1]
        mov     edi,[BUFFER]
        add     edi,eax
```

Hitung offset titik X1,Y1 dan tambahkan dengan alamat virtual screen.

```
        mov     al,[COL]
```

Isi register al dengan warna garis.

```
        rep     stosb
```

Lakukan pengisian titik sepanjang garis horizontal sejumlah ecx.

```
@HL_Vin:
        mov     esp,ebp
        pop     ebp

        ret
        ENDP
```

Fungsi VESA_Vline

Fungsi ini digunakan untuk menggambar garis lurus vertikal pada virtual screen.

```
VESA_VLine              PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,
```

```
Y2:DWORD,COL:BYTE
```

Parameter yang dikirimkan ialah BUFFER yaitu alamat memory virtual screen yang akan digambari. X1,Y1 yaitu koordinat titik pertama garis, dan Y2 yaitu koordinat Y titik kedua. COL ialah warna garis.

```
mov     eax,[Y1]
cmp     eax,[Y2]
jle     @ABNSwap
mov     ebx,[Y2]
mov     [Y1],ebx
mov     [Y2],eax
```

Deteksi, apakah Y1 lebih kecil dari Y2, jika tidak swap harga Y1 dan Y2.

```
@ABNSwap:

mov     eax,[Y1]
cmp     eax,0
jge     @VL_NoAddY
mov     [Y1],0
```

Deteksi, apakah garis berpotongan dengan atas layar. Jika ya, potong garis terhadap koordinat Y = 0.

```
@VL_NoAddY:
mov     eax,[Y2]
cmp     eax,[S_W]
jle     @VL_NoAddY2
mov     eax,[S_W]
mov     [Y2],eax
```

Deteksi, apakah garis berpotongan dengan bawah layar. Jika ya, potong garis terhadap koordinat Y = S_W.

```
@VL_NoAddY2:
mov     eax,[X1]
cmp     eax,0
jl      @VL_Vin
cmp     eax,[S_L]
jg      @VL_Vin
```

Deteksi, apakah garis berada di luar layar. Jika ya, batalkan penggambaran garis.

```
mov     eax,[Y2]
sub     eax,[Y1]
cmp     eax,0
```

```
        jl      @VL_Vin
        inc     eax
        mov     ecx,eax
```

Hitung panjang garis dan masukkan ke dalam register ecx. Jika panjang garis lebih kecil dari 0 maka batalkan penggambaran.

```
        mov     eax,[Y1]
        mul     [A_SL]
        add     eax,[X1]
        mov     edi,[BUFFER]
        add     edi,eax
```

Hitung offset titik pertama garis dan tambahkan dengan alamat memory virtual screen.

```
        mov     dl,[COL]
```

Isikan warna ke dalam register dl.

```
        @VL_Loop:
          mov     [edi],dl
          add     edi,[A_SL]
```

Isikan titik satu per satu ke dalam alamat memory dan tambahkan edi dengan A_SL untuk mengakses koordinat Y selanjutnya. Di sini tidak dapat dilakukan blok transfer dengan menggunakan rep stosb, karena titik-titik pada garis vertikal tidak menempati alamat memory yang besebelahan seperti halnya pada garis horizontal.

```
        loop @VL_Loop
```

Loop hingga seluruh garis telah digambar.

```
@VL_Vin:
        mov     esp,ebp
        pop     ebp

        ret

        ENDP
```

Fungsi VESA_XHLine

Fungsi ini digunakan untuk menggambar garis horizontal tapi dengan mode transparan, sehingga gambar di bawah garis tersebut masih dapat terlihat. Hal ini sering digunakan untuk efek-efek, juga untuk penggambaran kursor dari mouse. Kenapa? Karena kursor mouse

harus selalu terlihat jelas, sedangkan ada kemungkinan warna kursor mouse tersebut kebetulan sama dengan warna gambar di sekitarnya, sehingga kita tidak dapat melihatnya. Dengan mode transparansi, kita selalu dapat mengenalinya karena warna yang dipergunakan selalu lain dari warna yang ada di bawahnya.

Adapun cara kerja fungsi ini sama persis dengan fungsi VESA_Hline hanya saja terdapat perbedaan pada bagian peletakkan titiknya.

```
VESA_XHLine             PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,
                X2:DWORD
```

Karena dilakukannya mode transparansi, maka tidak diperlukan parameter COL yang berisi warna garis. Warna titik-titik garis akan dihitung dari warna titik-titik di bawahnya.

```
        mov     eax,[X1]
        cmp     eax,[X2]
        jle     @CANSwap
        mov     ebx,[X2]
        mov     [X1],ebx
        mov     [X2],eax
      @CANSwap:
```

Lakukan swapping antara X1 dan X2.

```
        mov     eax,[X1]
        cmp     eax,0
        jge     @XHL_NoAddX
        mov     [X1],0
      @XHL_NoAddX:
        mov     eax,[X2]
        cmp     eax,S_L
        jle     @XHL_NoAddX2
        mov     eax,[S_L]
        mov     [X2],eax
      @XHL_NoAddX2:
```

Potong garis terhadap batas layar.

```
        mov     eax,[Y1]
        cmp     eax,0
        jl      @XHL_Vin
        cmp     eax,S_W
        jg      @XHL_Vin
```

Deteksi, apakah garis di luar layar.

```
        mov     eax,[X2]
        sub     eax,[X1]
        cmp     eax,0
        jl      @XHL_Vin
        inc     eax
        mov     ecx,eax
```

Hitung panjang garis.

```
        mov     eax,[Y1]
        mul     [A_SL]
        add     eax,[X1]
        mov     edi,[BUFFER]
        add     edi,eax
```

Hitung offset garis.

```
      @XHLLoop:
        xor     [edi],0fh
        inc     edi
```

Lakukan fungsi XOR terhadap data yang telah ada di layar (titik di bawah garis) dengan 0fh sehingga 4 bit pertama dari data dibalik (1011 binary menjadi 0100 binary). Kemudian tambahkan edi untuk memproses koordinat X selanjutnya.

```
      loop @XHLLoop
```

Loop hingga seluruh garis selesai digambar.

```
@XHL_Vin:
        mov     esp,ebp
        pop     ebp

        ret
        ENDP
```

Fungsi VESA_XVLine

Fungsi ini digunakan untuk menggambar garis vertikal dengan mode transparan.

Implementasinya juga hampir sama dengan fungsi VESA_Vline hanya terdapat perbedaan di bagian peletakan titiknya.

```
VESA_XVLine             PROC NEAR
```

```
push    ebp
mov     ebp,esp
ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,  Y2:DWORD

mov     eax,[Y1]
cmp     eax,[Y2]
jle     @CBNSwap
mov     ebx,[Y2]
mov     [Y1],ebx
mov     [Y2],eax
```

Lakukan swapping antara Y1 dan Y2.

```
@CBNSwap:

  mov     eax,[Y1]
  cmp     eax,0
  jge     @XVL_NoAddY
  mov     [Y1],0
@XVL_NoAddY:
  mov     eax,[Y2]
  cmp     eax,[S_W]
  jle     @XVL_NoAddY2
  mov     eax,[S_W]
  mov     [Y2],eax
@XVL_NoAddY2:
```

Potong garis terhadap batas layar.

```
  mov     eax,[X1]
  cmp     eax,0
  jl      @XVL_Vin
  cmp     eax,[S_L]
  jg      @XVL_Vin
```

Deteksi, apakah garis berada di luar layar.

```
  mov     eax,[Y2]
  sub     eax,[Y1]
  cmp     eax,0
  jl      @XVL_Vin
  inc     eax
  mov     ecx,eax
```

Hitung panjang garis dan masukkan ke dalam ecx sebagai counter. Jika panjang garis lebih kecil dari 0 maka batalkan proses penggambaran.

```
  mov     eax,[Y1]
```

```
        mul     [A_SL]
        add     eax,[X1]
        mov     edi,[BUFFER]
        add     edi,eax
```

Hitung offset dari titik pertama garis.

```
        @XVL_Loop:
          xor     [edi],0fh
          add     edi,[A_SL]
```

Balik 4 bit pertama dari titik yang sedang diproses sehingga diperoleh efek transparan. Kemudian tambahakan edi dengan A_SL untuk mengakses koordinat Y selanjutnya.

```
        loop @XVL_Loop
```

Loop hingga seluruh bagian garis selesai diproses.

```
@XVL_Vin:
        mov     esp,ebp
        pop     ebp
        ret

        ENDP
```

Fungsi VESA_VWP_Hline

Fungsi ini adalah mode viewport dari fungsi VESA_Hline.

```
VESA_VWP_HLine              PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,
                X2:DWORD,COL:BYTE

        mov     eax,[X1]
        cmp     eax,[X2]
        jle     @BANSwap
        mov     ebx,[X2]
        mov     [X1],ebx
        mov     [X2],eax
      @BANSwap:

        mov     eax,[X1]
        cmp     eax,[V_X1]
        jge     @VWPHL_NoAddX
```

```
        mov     eax,[V_X1]
        mov     [X1],eax
      @VWPHL_NoAddX:
        mov     eax,[X2]
        cmp     eax,[V_X2]
        jle     @VWPHL_NoAddX2
        mov     eax,[V_X2]
        mov     [X2],eax
      @VWPHL_NoAddX2:
        mov     eax,[Y1]
        cmp     eax,[V_Y1]
        jl      @VWPHL_Vin
        cmp     eax,[V_Y2]
        jg      @VWPHL_Vin
```

Pemotongan garis dilakukan terhadap koordinat viewport bukannya terhadap koordinat minimum-maksimum layar.

```
        mov     eax,[X2]
        sub     eax,[X1]
        cmp     eax,0
        jl      @VWPHL_Vin
        inc     eax
        mov     ecx,eax

        mov     eax,[Y1]
        mul     [A_SL]
        add     eax,[X1]
        mov     edi,[BUFFER]
        add     edi,eax
        mov     al,[COL]
        rep     stosb

@VWPHL_Vin:
        mov     esp,ebp
        pop     ebp
        ret
        ENDP
```

Fungsi VESA_VWP_Vline

Fungsi ini merupakan mode viewport dari fungsi VESA_Vline.

```
VESA_VWP_VLine              PROC NEAR

        push    ebp
        mov     ebp,esp
        ARG     BUFFER:DWORD, X1:DWORD, Y1:DWORD,
```

```
Y2:DWORD,COL:BYTE
    mov     eax,[Y1]
    cmp     eax,[Y2]
    jle     @BBNSwap
    mov     ebx,[Y2]
    mov     [Y1],ebx
    mov     [Y2],eax
  @BBNSwap:

    mov     eax,[Y1]
    cmp     eax,[V_Y1]
    jge     @VWPVL_NoAddY
    mov     eax,[V_Y1]
    mov     [Y1],eax
  @VWPVL_NoAddY:
    mov     eax,[Y2]
    cmp     eax,[V_Y2]
    jle     @VWPVL_NoAddY2
    mov     eax,[V_Y2]
    mov     [Y2],eax
  @VWPVL_NoAddY2:
    mov     eax,[X1]
    cmp     eax,[V_X1]
    jl      @VWPVL_Vin
    cmp     eax,[V_X2]
    jg      @VWPVL_Vin
```

Pemotongan dilakukan terhadap koordinat viewport.

```
    mov     eax,[Y2]
    sub     eax,[Y1]
    cmp     eax,0
    jl      @VWPVL_Vin
    inc     eax
    mov     ecx,eax

    mov     eax,[Y1]
    mul     [A_SL]
    add     eax,[X1]
    mov     edi,[BUFFER]
    add     edi,eax
    mov     dl,[COL]
  @VWPVL_Loop:
      mov    [edi],dl
      add    edi,[A_SL]
    loop @VWPVL_Loop

@VWPVL_Vin:
```

```
mov     esp,ebp
pop     ebp
ret

ENDP
```

Fungsi VESA_VWP_XHLine

Fungsi ini merupakan mode viewport dari fungsi VESA_XHLine. Seperti sebelumnya, implementasinya sama persis hanya terdapat perbedaan pada pemotongan garis. Kita tidak akan membahas listing programnya agar buku ini tidak terlalu tebal ☺.

Fungsi VESA_VWP_XVLine

Fungsi ini merupakan mode viewport dari fungsi VESA_XVLine. Seperti sebelumnya, implementasinya sama juga sehingga kita tidak akan membahas listing programnya.

Fungsi VESA_Line

Fungsi ini digunakan untuk menggambar garis bebas pada virtual screen. Jangan kaget ya, fungsi ini sangat panjang karena banyak situasi-situasi yang harus diatasi secara terpisah. Tetapi karena fungsi ini merupakan salah satu fungsi yang penting, maka kita akan membahasnya secara detil.

Pembuatan garis ini dilakukan dengan metode bresenham yang dibantu dengan fixed point algorithm. Keduanya telah kita bahas pada seri-seri Pemrograman Animasi dan Game Profesional sebelumnya.

Catatan

Metode bresenham ialah metode untuk menggambar titik-titik pada garis dengan memilih titik terdekat dari titik-titik tersebut sebagai perkiraan, karena layar monitor tidak seperti kertas gambar di mana kita dapat meletakkan titik di mana saja dan dengan ukuran berapa saja.

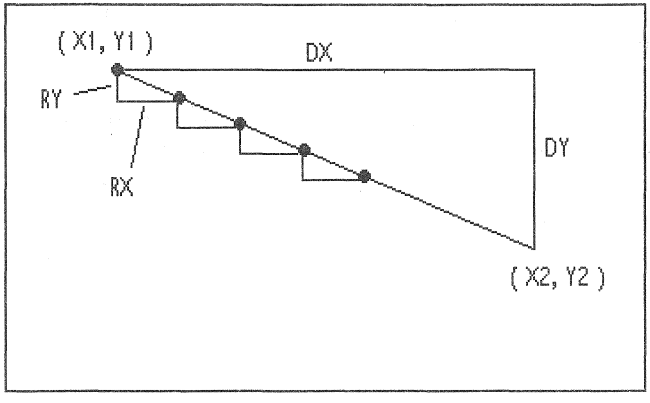
Catatan

Metode fixed point algorithm ialah metode menghitung bilangan pecahan dengan menggunakan bilangan desimal yang dikalikan dengan bilangan tertentu sebagai angka 1. Misalnya angka pengalinya ialah 256 maka bilangan 256 dianggap sebagai angka 1, sedangkan bilangan 128 dianggap sebagai angka 0.5 dan sebagainya, sehingga

memungkinkan kita merepresentasikan bilangan pecahan ke dalam bilangan desimal. Hal ini dilakukan karena register dalam bahasa assembler tidak mampu menampung bilangan pecahan. Sebenarnya dengan floating point, kita dapat menghitung bilangan pecahan tetapi fasilitas ini hanya ada pada komputer yang memiliki math-coprocessor (untuk komputer jenis 386) sehingga topiknya tidak akan kita bahas.

Sebenarnya proses penggambaran garisnya sendiri cukup singkat, hanya saja proses pemotongan garis tersebut terhadap batas layarlah yang menyebabkan fungsi ini menjadi membengkak.

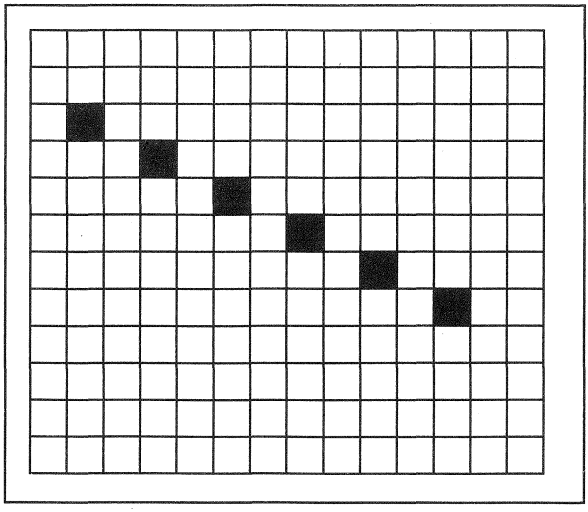
Untuk menggambar garis dari sebuah titik ke titik lain, kita harus meletakkan titik-titik yang menghubungkan antara titik yang pertama tadi dengan titik yang kedua. Caranya ialah dengan melakukan loop dengan dua buah variabel yaitu RX dan RY. RX adalah variabel yang menentukan lompatan horizontal untuk setiap titik, sedangkan RY adalah variabel yang menentukan lompatan vertikal untuk setiap titik. Perhatikan Gambar 5.3.



Gambar 5.3 Penggambaran Garis dengan Metode Bresenham

Sekarang yang menjadi masalah ialah berapa jumlah titik yang berada di antara garis tersebut? Dengan mengetahui jumlah titik ini, kita akan dapat menghitung berapa RX dan RY antara titik. Masalahnya ialah, kita harus mengambil jumlah titik ini seminimal mungkin tetapi dengan jumlah ini, garis harus masih terlihat lurus (tidak terputus-putus).

Kenapa dengan jumlah titik yang sedikit, garis terlihat putus-putus? Coba perhatikan Gambar 5.4 dan Gambar 5.5 yang memperlihatkan sebuah garis yang diperbesar agar terlihat lebih jelas.



Gambar 5.4 Garis dengan Jumlah Titik 6

Perhatikan bahwa garis yang menggunakan jumlah titik 11 terlihat lebih realistis.

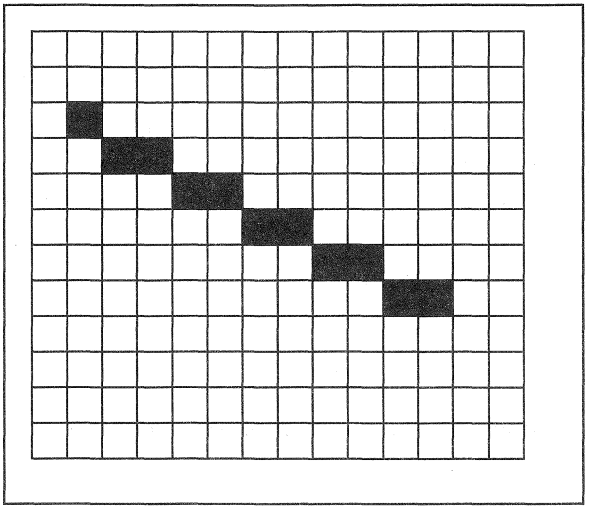
Bagaimana cara menghitung jumlah titik yang paling cocok untuk sebuah garis? Caranya ialah dengan melihat panjang garis secara horizontal (dinamakan DX) dan panjang garis secara vertikal (dinamakan DY). Harga yang paling besar di antara DX dan DY-lah yang menjadi perkiraan patokan jumlah titik antara suatu garis. Coba perhatikan Gambar 5.5, di sana terlihat DX = 11 sedangkan DY = 6. Dan terbukti dengan jumlah titik = DX = 11 maka garis terlihat realistis.

Sekarang bagaimana cara kita menghitung RX dan RY? RX dan RY dihitung dengan metode yang berbeda dalam dua situasi yang berbeda. Pada situasi pertama jika DX lebih besar dari DY maka RX = 1 sedangkan RY = DY/DX. Jika DY lebih besar dari DX maka digunakan situasi kedua yaitu RY = 1 sedangkan RX = DX/DY.

Coba kita praktekkan pada garis dalam Gambar 5.5. DX = 11 dan DY = 6. Karena DX lebih besar dari DY maka digunakan situasi pertama yaitu RX = 1 dan RY = DY/DX = 6/11 = 0.55.

Karena RX dan RY ini membutuhkan variabel yang dapat menampung bilangan pecahan, maka kita harus menerapkan metode fixed point pada kedua variabel ini. Adapun fixed point yang akan kita gunakan ialah bilangan 65536 sebagai angka 1 (akurasi 16 bit). Karena kita bermain dengan resolusi tinggi, maka kita membutuhkan akurasi

yang lebih besar dibandingkan akurasi 8 bit yang kita gunakan pada seri sebelumnya dalam perakitan PUT3D di VGACAD.TPU.



Gambar 5.5 Garis dengan Jumlah Titik 11

Langkah selanjutnya ialah dilakukan loop sejumlah titik yang ada dalam garis dan pada setiap loopnya, dua buah variabel yaitu Xfixed dan Yfixed (kedua variabel ini diinisialisasi dengan harga X1 dan Y1) akan selalu diakumulasi dengan RX dan RY yang menunjukkan koordinat titik yang sedang digambar. Tentu saja kedua variabel ini juga menggunakan metode fixed point.

Sekarang mari kita lihat secara langsung perakitan fungsi VESA_Line ini.

```
VESA_Line       PROC NEAR
      LOCAL     IFlag:DWORD,CalX:DWORD,CalY:DWORD,DelX:DWORD,

DelY:DWORD,Dsx:DWORD,Dsy:DWORD,Rx:DWORD,Ry:DWORD,
                XFixed:DWORD,YFixed:DWORD,TLoop:DWORD=AUTO_SIZE
```

Karena fungsi ini cukup rumit, maka diperlukan variabel-variabel lokal.

```
      push      ebp
      mov       ebp,esp
      sub       esp,AUTO_SIZE
```

Kurangi esp (stack pointer) dengan AUTO_SIZE (AUTO_SIZE menunjukkan alamat memory setelah variabel lokal) sehingga stack

pointer menunjuk pada alamat memory di atas variabel lokal. Hal ini dilakukan agar stack tidak bertubrukan dengan lokasi di mana variabel lokal di simpan dalam memory.

```
ARG         PICBUFF:DWORD,X1:DWORD,Y1:DWORD,X2:DWORD,
            Y2:DWORD,COL:BYTE
```

Parameter yang dikirimkan ialah PICBUFF yaitu alamat virtual screen yang akan digambari. (X1, Y1) adalah koordinat awal garis dan (X2,Y2) adalah koordinat akhir garis. COL ialah warna garis yang akan digambar.

```
mov         esi,[PICBUFF]
mov         ebx,[Y2]
sub         ebx,[Y1]
jge         @VESLno_swap
```

Sama seperti sebelumnya, di sini dilakukan juga swapping antara Y1 dan Y2 jika Y2 lebih kecil dari Y1.

```
mov         ecx,[Y2]
mov         ebx,[Y1]
mov         [Y2],ebx
mov         [Y1],ecx
```

Tukar Y1 dan Y2.

```
sub         ebx,ecx
mov         [Dsy],ebx
```

Hitung DY dan simpan ke dalam Dsy.

```
mov         ecx,[X2]
mov         ebx,[X1]
mov         [X2],ebx
mov         [X1],ecx
```

Karena Y1 ditukar dengan Y2 maka tukar juga X1 dan X2. Jika tidak maka kita akan menghasilkan pasangan koordinat yang salah yaitu (X1,Y2)-(X2,Y1).

```
  jmp       @VESLNo_Add
@VESLno_swap:
  mov       [Dsy],ebx
@VESLNo_Add:
  mov       eax,[X1]
  sub       eax,[X2]
  push      eax
```

Hitung selisih antara X1 dan X2 dan simpan dalam stack. Selisih ini nantinya akan digunakan sebagai tanda arah garis. Jika selisih ini berharga negatif maka arah garis ialah dari kiri ke kanan dan sebaliknya jika harganya positif.

```
cmp         eax,0
jge         @VESL_Case2
neg         eax
```

Selisih antara X1 dan X2 tadi digunakan sebagai DX. Jika selisih ini negatif maka positifkan dengan instruksi neg.

```
@VESL_Case2:
  mov       [Dsx],eax
```

Simpan DX dalam Dsx.

Langkah selanjutnya ialah menganalisis antara dua situasi (DX atau DY yang lebih besar).

```
mov         eax,[Dsx]
sub         eax,[Dsy]
jge         @VESXFixed
```

Lihat antara Dsx dan Dsy. Jika Dsx lebih besar maka lompat ke VESXFixed yang berarti situasi pertama. Sedangkan proses di bawah ini untuk situasi kedua di mana DY lebih besar dari DX.

```
mov         [Ry],10000h
```

Tetapkan RY = 1 (dalam fixed point 16 bit, 1 = 65536 desimal = 10000 hex).

```
mov         eax,[Dsx]
shl         eax,16
```

Kalikan DX dengan 65536 untuk menciptakan bilangan fixed point dengan angka 1 = 65536.

```
xor         edx,edx
div         [Dsy]
```

Bagi dengan DY.

```
mov         [Rx],eax
```

RX = DX/DY di simpan dalam variabel Rx.

```
mov         eax,[Dsy]
jmp         @VESTest_Crop
```

Lompat ke VESTest_Crop untuk melakukan pemotongan garis terhadap batas layar.

Proses di bawah ini dilakukan untuk situasi pertama di mana DX lebih besar dari DY.

```
@VESXFixed:
  mov       [Rx],10000h
```

Tetapkan harga RX = 1 (dalam fixed point 16 bit, 1 = 65536 desimal = 10000 hex).

```
  mov       eax,[Dsy]
  shl       eax,16
  xor       edx,edx
  div       [Dsx]
  mov       [Ry],eax
```

RY = DY/DX dan simpan dalam variabel RY.

```
  mov       eax,[Dsx]
@VESTest_Crop:
  mov      [IFlag],0
```

Iflag digunakan untuk menyimpan informasi apakah garis terpotong batas layar atau tidak. Pertama-tama diisi dengan angka 0 yang berarti tidak memotong batas layar.

```
  mov       [TLoop],eax
```

Isikan Tloop yang menunjukkan jumlah titik antara dua garis dengan register eax, register eax ini telah diisi dengan DX/DY tergantung dengan situasi pertama/kedua.

```
  mov       [XFixed],0
  mov       [YFixed],0
```

Kosongkan variabel Xfixed dan Yfixed.

```
  mov       eax,[S_L]
  cmp       [X1],eax
  jle       @VESNext1
```

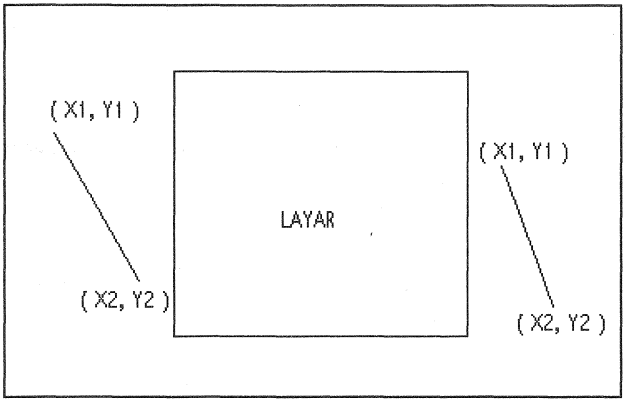
Kalau X1 berada di sebelah kanan layar.

```
  cmp       [X2],eax
  jle       @VESNext1
  jmp       @VESFinish
```

Dan juga X2 berada di sebelah kanan layar maka garis berada di luar layar, karena kedua titik penghubungnya berada di sebelah kanan layar. Batalkan proses penggambaran.

```
@VESNext1:
  cmp       [X1],0
  jge       @VESNext2
  cmp       [X2],0
  jge       @VESNext2
  jmp       @VESFinish
```

Jika kedua titik penghubungnya berada di sebelah kiri layar, batalkan proses penggambaran. Situasi ini digambarkan pada Gambar 5.6



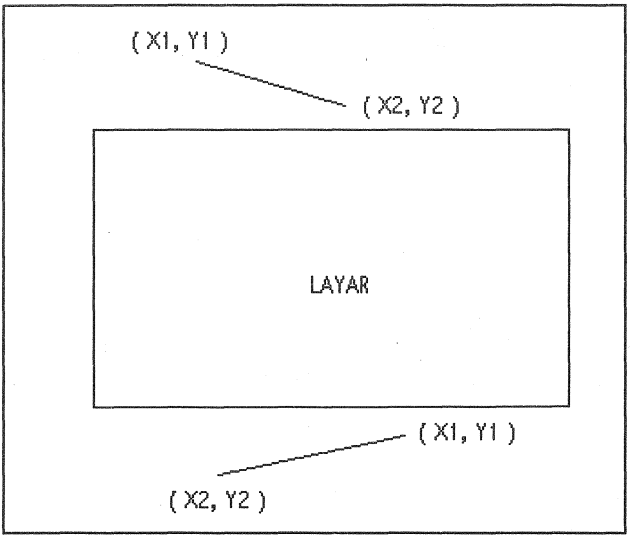
Gambar 5.6 Kedua Garis Berada di Sebelah Kiri dan Kanan Layar

```
@VESNext2:
  mov       eax,[S_W]
  cmp       [Y1],eax
  jg        @VESFinish
  cmp       [Y2],0
  jl        @VESFinish
```

Jika koordinat Y1 (koordinat atas garis) berada di bawah batas bawah layar, maka garis berada di luar layar. Juga jika koordinat Y2 (koordinat bawah garis) berada di atas batas atas layar, maka garis berada di luar layar. Situasi ini digambarkan pada Gambar 5.7.

Proses-proses di bawah ini dilakukan untuk memotong garis terhadap batas-batas layar.

```
  mov       eax,[Y1]
  neg       eax
  jle       @VESTest_Side
```



Gambar 5.7 Kedua Garis yang Berada di Atas dan Bawah Layar

Jika garis tidak memotong atas layar, lompat ke VESTest_Side untuk mendeteksi perpotongan dengan batas samping layar.

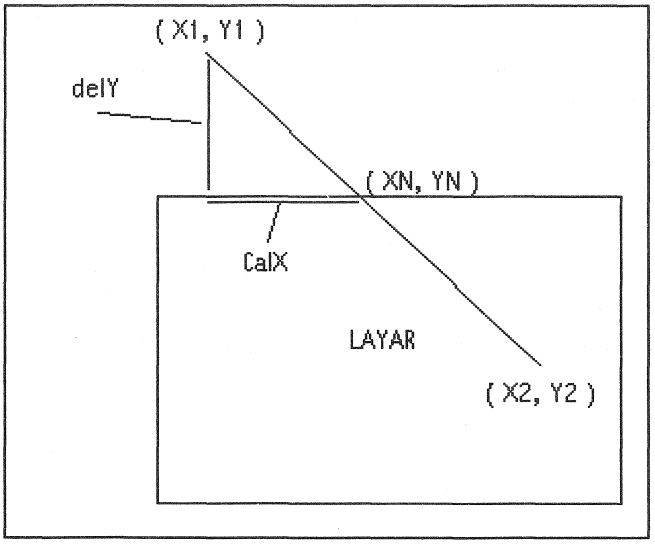
Proses di bawah ini dilakukan untuk mencari koordinat baru (XN,YN) dari garis yang memotong atas layar.

```
mov     [delY],eax
mul     [Rx]
xor     edx,edx
div     [Ry]
mov     [CalX],eax
```

Kita tidak perlu menghitung YN karena garis yang akan dipotong terhadap batas atas layar akan memiliki koordinat Y = 0. Yang perlu dihitung ialah koordinat X-nya. Caranya ialah dengan memanfaatkan RX dan DY yang merupakan counter dari posisi horizontal dan vertikal. Dengan memanfaatkan jarak seberapa jauh posisi vertikal menerobos batas layar, kita dapat menghitung panjang horizontal dari bagian garis yang menerobos batas layar:

CalX = (delY*RX)/RY

Perhatikan Gambar 5.8. CalX di sini hanyalah panjang horizontal dari bagian garis yang menerobos batas atas layar. Sehingga untuk mendapatkan koordinat X yang baru, kita tinggal menambahkan panjang ini dengan koordinat X sebelum dipotong.



Gambar 5.8 Pemotongan Garis yang Menerobos Batas Atas Layar

```
pop         ecx
push        ecx
cmp         ecx,0
jle         @VESNo_neg
neg         eax
```

Jika arah garis dari kiri ke kanan (arah garis dapat dilihat dari selisih X2 dan X1 yang disimpan dalam stack) maka penghitungan koordinat XN akan dilakukan dengan menambah koordinat X1 dengan CalX dan jika arahnya sebaliknya, maka dilakukan pengurangan antara X1 dengan CalX yang dilakukan pada satu baris di bawah ini (add eax, [X1]).

```
@VESNo_neg:
  add       eax,[X1]
  cmp       eax,[S_L]
  jg        @VESTest_Side
  cmp       eax,0
  jl        @VESTest_Side
```

Selanjutnya dideteksi, apakah koordinat garis yang baru ini memotong batas samping layar? Jika ya, lompat ke VESTest_Side untuk melakukan pemotongan terhadap samping layar.

```
mov         [IFlag],1
```

Karena garis telah memotong atas layar maka set Iflag menjadi 1.

Proses selanjutnya ialah mengurangi jumlah titik yang terletak di antara garis karena telah dilakukan pemotongan sehingga harga ini berkurang.

```
cmp         [Rx],10000h
jne         @VESUse_Ry
```

Mula-mula lihat dulu apakah situasi yang digunakan ialah nomor satu atau nomor dua.

Proses di bawah ini ialah situasi nomor satu di mana DX = Jumlah titik.

```
mov         eax,[CalX]
mov         ebx,eax
sub         [TLoop],eax
```

Karena pada situasi ini, lompatan horizontal RX = 1 dan jumlah titik = DX maka secara langsung jumlah pengurangan titik ialah sama dengan panjang horizontal yang telah terpotong yaitu CalX. Kurangi Tloop dengan CalX.

```
shl         eax,16
mov         [XFixed],eax
```

Langkah selanjutnya ialah mengisi Xfixed dengan X1 yang telah terpotong dan harus dalam mode fixed point. Caranya cukup mengalikan X1 dengan 65536 (shl 16 = perkalian dengan 65536).

```
mov         eax,ebx
mul         [Ry]
mov         [YFixed],eax
```

Kemudian Yfixed juga diisi dengan Y1 yang telah terpotong dalam format Fixed point.

```
jmp         @VESTest_Bottom
```

Lompat ke bagian pemotongan batas bawah layar.

```
@VESUse_Ry:
  mov       eax,[delY]
  mov       ebx,eax
  sub       [TLoop],eax
  shl       eax,16
  mov       [YFixed],eax
  mov       eax,ebx
```

```
mul         [Rx]
mov         [XFixed],eax
```

Pada situasi kedua di mana lompatan vertikal RY = 1 dan jumlah titik = DY sehingga kita cukup mengurangi Tloop dengan delY. Kemudian isikan ke Yfixed dalam format fixed point. Untuk menghitung Xfixed, dilakukan dengan mengalikan delY dengan Rx. Yfixed dan Xfixed adalah koordinat dari titik baru setelah dipotong dalam format Fixed Point.

```
jmp         @VESTest_Bottom
```

Lompat ke bagian pemotongan batas bawah layar.

```
@VESTest_Side:
  pop       ecx
  push      ecx
  cmp       ecx,0
  jl        @VESTo_Right
```

Periksa arah garis. Jika dari kiri ke kanan maka lompat ke VESTo_Right.

Proses di bawah ini akan memproses garis yang arahnya dari kanan ke kiri.

```
mov         eax,[X1]
sub         eax,[S_L]
jle         @VESTest_Bottom
```

Jika garis tidak memotong batas kanan layar, lompat ke bagian pemotongan bawah layar.

```
mov         [delX],eax
jmp         @VESCont_Side
```

Garis memotong batas kanan layar, hasil pengurangan di atas adalah delX. Lompat ke bagian pemotongan samping.

Program di bawah ini ialah bagian yang mengatasi garis dari kiri ke kanan.

```
@VESTo_Right:
  mov       eax,[X1]
  neg       eax
  jle       @VESTest_Bottom
  mov       [delX],eax
```

Jika garis memotong kiri layar maka delX dihitung dari kelebihan garis dari kiri layar. Jika tidak, lompat ke bagian pemotongan bawah layar.

```
@VESCont_Side:
  mul       [Ry]
  xor       edx,edx
  div       [Rx]
  mov       [CalY],eax
```

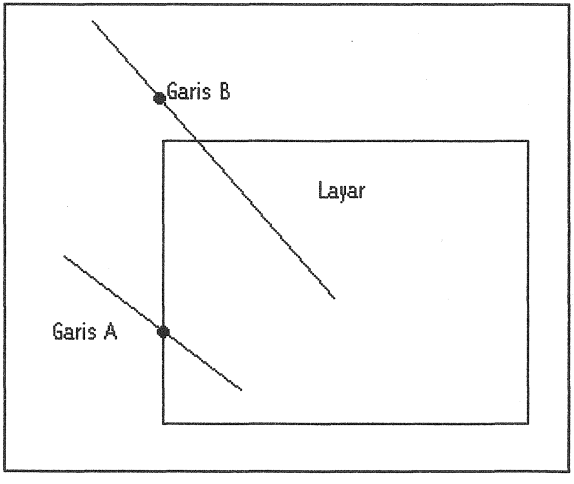
Hitung CalY yang diperoleh dari delX * Ry/Rx. CalY ini ialah pemotongan panjang vertikal garis.

```
  add       eax,[Y1]
```

Tambahkan ke Y1 untuk mendapatkan koordinat Y yang baru setelah dipotong.

```
  cmp       eax,0
  jl        @VESTest_Bottom
  cmp       eax,[S_L]
  jg        @VESTest_Bottom
```

Deteksi, apakah koordinat Y yang baru setelah dipotong benar-benar memotong pinggiran kiri layar. Maksudnya ialah hasil pemotongan seperti garis A pada Gambar 5.9.



Gambar 5.9 Jenis-jenis Perpotongan Terhadap Sisi Layar

Garis A ini setelah dipotong terhadap sisi kiri layar, letak titik baru hasil pemotongan masih berada pada sisi kiri layar. Situasi inilah

yang disebut benar-benar memotong sisi kiri layar. Pada garis B, setelah dipotong terhadap kiri layar, titik baru hasil pemotongan berada jauh di atas sisi kiri layar. Situasi ini disebut tidak benar-benar memotong sisi kiri layar. Hal ini harus dilakukan pemotongan lebih lanjut yaitu terhadap sisi atas layar.

```
        mov         [IFlag],1
        cmp         [Rx],10000h
```

Tandai Iflag sebagai 1 yang berarti garis memotong sisi kiri layar.

```
        jne         @VESUse_RyS
```

Proses-proses selanjutnya ialah hampir sama dengan proses pemotongan bagian atas garis. Hanya bedanya, di sini akan digunakan delX bukannya delY.

```
        mov         eax,[delX]
        sub         [TLoop],eax
        shl         eax,16
        mov         [XFixed],eax
        mov         eax,[delX]
        mul         [Ry]
        mov         [YFixed],eax
        jmp         @VESTest_Bottom
    @VESUse_RyS:
```

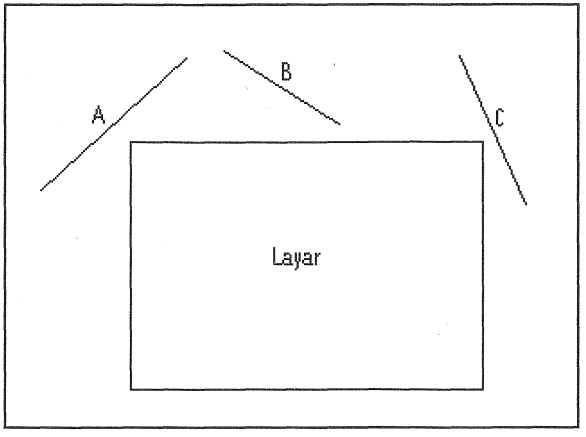
Situasi nomor satu. Gunakan delX sebagai pengurang jumlah titik dan isikan Xfixed dan Yfixed.

```
        mov         eax,[CalY]
        sub         [TLoop],eax
        shl         eax,16
        mov         [YFixed],eax
        mov         eax,[CalY]
        mul         [Rx]
        mov         [XFixed],eax
```

Situasi nomor dua. Gunakan CalY untuk menghitung pemotongan jumlah titik. Isikan Xfixed dan Yfixed.

```
    @VESTest_Bottom:
```

Proses pendeteksian lebih lanjut ialah memeriksa apakah garis tidak memotong batas layar (Iflag = 0) tetapi titik pertama dari garis berada di luar layar. Situasi ini dapat digambarkan pada beberapa garis dalam Gambar 5.10.



Gambar 5.10 Garis A, B dan C Berada di Luar Layar

```
cmp         [IFlag],0
jne         @VESTest_Bottom_Crop
mov         eax,[S_L]
cmp         [X1],eax
jg          @VESFinish
cmp         [X1],0
jl          @VESFinish
cmp         [Y1],0
jl          @VESFinish
mov         eax,[S_W]
cmp         [Y1],eax
jg          @VESFinish
```

Jika garis tidak memotong batas layar sama sekali dan titik pertama dari garis berada di luar layar, maka garis benar-benar berada di luar layar sehingga proses penggambaran dibatalkan.

Proses selanjutnya ialah hampir sama seperti seluruh proses pemotongan di atas, hanya saja di sini dilakukan terhadap bagian bawah garis.

Untuk mempersingkat pembahasan, kita akan membahas langkah selanjutnya secara singkat hingga sampai pada bagian penggambaran garis.

```
@VESTest_Bottom_Crop:
  mov       eax,[Y2]
  sub       eax,[S_W]
  jle       @VESTest_Final
  mov       [delY],eax
  mul       [Rx]
```

```
    xor         edx,edx
    div         [Ry]
    mov         [CalX],eax

    pop         ecx
    push        ecx
    cmp         ecx,0
    jge         @VESBot_Add
    neg         eax
@VESBot_Add:
    add         eax,[X2]
    cmp         eax,[S_L]
    jg          @VESTest_Final
    cmp         eax,0
    jl          @VESTest_Final
```

Sampai pada bagian ini, garis benar-benar memotong bawah layar. Karena setelah dipotong terhadap bawah layar, titik hasil pemotongan berada pada sisi bawah layar. Situasi yang sama dengan yang digambarkan pada Gambar 5.9.

```
    cmp         [Rx],10000h
    jne         @VESUse_RyB
    mov         eax,[CalX]
```

Gunakan CalX sebagai pemotong karena situasi yang terjadi ialah situasi pertama.

```
    jmp         @VESCont_Bottom
@VESUse_RyB:
```

Gunakan delY sebagai pemotong karena situasi yang terjadi ialah situasi kedua.

```
    mov         eax,[delY]

@VESCont_Bottom:
    sub         [TLoop],eax
    jmp         @VESTest_Dir


@VESTest_Final:
```

Deteksi, apakah bagian bawah garis memotong sisi kiri kanan layar.

```
    pop         ecx
    push        ecx
    cmp         ecx,0
```

```
        jge         @VESTo_RightF
```

Periksa arah garis (dari kiri ke kanan atau kanan ke kiri).

Proses di bawah ini ialah untuk menangani garis dari kanan ke kiri.

```
        mov         eax,[X2]
        sub         eax,[S_L]
        jle         @VESTest_Dir
```

Jika bagian akhir garis tidak memotong kanan layar, maka lompat ke bagian penggambaran garis.

```
        mov         [delX],eax
        jmp         @VESCont_Final
```

Jika memotong lompat ke VESCont_Final, panjang pemotongan horizontal ialah delX.

```
    @VESTo_RightF:
```

Garis dengan arah dari kiri ke kanan.

```
        mov         eax,[X2]
        neg         eax
        jle         @VESTest_Dir
```

Garis tidak memotong kiri layar.

```
        mov         [delX],eax
```

Garis memotong kiri layar, pemotongan = delX.

```
    @VESCont_Final:
      mul         [Ry]
      xor         edx,edx
      div         [Rx]
      mov         [CalY],eax
      mov         ebx,[Y2]
      sub         ebx,eax
      cmp         ebx,0
      jl          @VESTest_Dir
      cmp         ebx,[S_L]
      jg          @VESTest_Dir

      mov         ebx,[delX]
      cmp         [Rx],10000h
      je          @VESUse_RxF
      mov         ebx,[CalY]
```

```
@VESUse_RxF:
  sub       [TLoop],ebx
```

Jika garis memotong sisi layar, kurangi Tloop.

```
@VESTest_Dir:
  pop       eax
  push      eax
  cmp       eax,0
  jl        @VESDraw_Line
```

Lihat arah garis.

```
  neg       [Rx]
  neg       [XFixed]
```

Jika dari kanan ke kiri, negatifkan harga Xfixed dan Rx.

```
@VESDraw_Line:
  mov       eax,[Y1]
  mul       [A_SL]
  add       eax,[X1]
  add       esi,eax
```

Hitung offset dari titik pertama garis.

```
  mov       ecx,[TLoop]
  inc       ecx
  cmp       ecx,0
```

Masukkan Tloop ke dalam ecx sebagai counter dari titik-titik dalam garis.

```
  jle        @VESFinish
@VESLLoop:
  mov       eax,[YFixed]
  shr       eax,16
  adc       eax,0
```

Konversi mode fixed point ke desimal biasa untuk menghitung offset dari titik. Penambahan dengan carry dilakukan untuk memperoleh akurasi 1 desimal di belakang koma. Karena dengan ini akan diperhitungkan 1 bit di sebelah bit 0.

```
  mul       [A_SL]
  mov       ebx,[XFixed]
  sar       ebx,16
  adc       ebx,0
```

Konversi Xfixed ke mode desimal biasa.

```
        add         eax,ebx
```

Tambahkan ebx ke eax untuk mendapatkan offset titik yang sedang diproses.

```
        mov         dl,[Col]
        mov         BYTE PTR [esi+eax],dl
```

Isikan warna titik ke virtual screen.

```
        mov         eax,[Rx]
        add         [XFixed],eax
        mov         eax,[Ry]
        add         [YFixed],eax
```

Tambahkan koordinat Xfixed dan Yfixed dengan Rx dan Ry untuk memproses titik selanjutnya.

```
    loop @VESLLoop
```

Loop hingga seluruh bagian garis telah diisi.

```
    @VESFinish:
      pop       eax
      mov       esp,ebp
      pop       ebp
      ret

VESA_Line       ENDP
```

Akhirnya, selesai juga penderitaan kita membahas fungsi VESA_Line di atas. Memang fungsi ini adalah fungsi yang paling sulit dibuatnya di antara fungsi-fungsi yang lain.

Fungsi selanjutnya yang akan kita lihat ialah VESA_Xline. Fungsi ini tidak diciptakan, tetapi fungsi Xline dengan mode viewport bernama VESA_VWP_Xline telah disediakan sehingga kita dapat menggunakannya. Hanya saja ingat karena digunakannya mode viewport, jika kita hendak menggunakannya tanpa mempedulikan viewport, set viewport ke (0,0)-(639,399) atau dengan tidak mengganti viewport sama sekali karena pada saat awal, viewport akan otomatis diset ke ukuran tersebut.

Tenang... kita tidak akan membahas fungsi ini secara detil, karena toh mirip dengan fungsi di atas, kita hanya akan melihat bagian transparansinya saja.

Penjelasan Potongan Fungsi VESA_VWP_Xline

```
@uVWPLLoop:
      mov       eax,[YFixed]
      shr       eax,16
      adc       eax,0
      mul       [A_SL]
      mov       ebx,[XFixed]
      sar       ebx,16
      adc       ebx,0
      add       eax,ebx
      mov       dl,BYTE PTR [esi+eax]
      xor       dl,0fh
      mov       BYTE PTR [esi+eax],dl
```

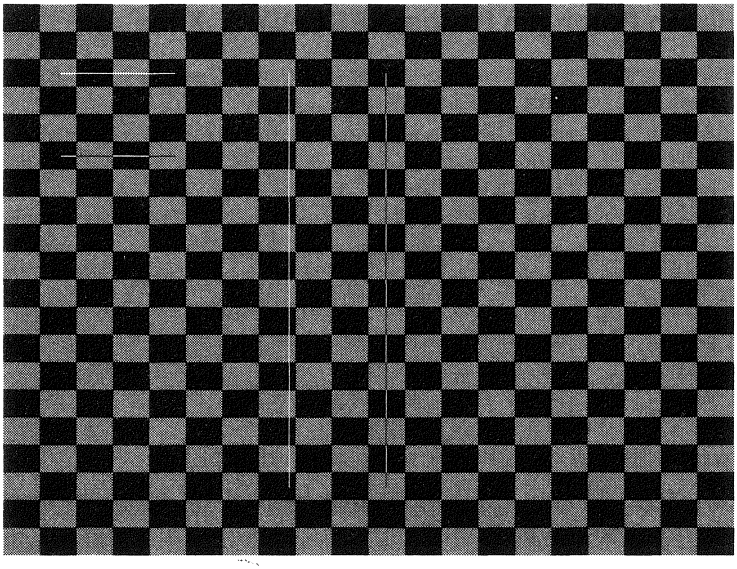
Baca warna titik di bawah titik yang akan diletakkan.

Lakukan XOR dengan 0fh untuk membalik 4 bit terkecil. Kemudian isikan kembali ke layar.

```
      mov       eax,[Rx]
      add       [XFixed],eax
      mov       eax,[Ry]
      add       [YFixed],eax
    loop @uVWPLLoop

    @uVWPFinish:
      pop       eax
      mov       esp,ebp
      pop       ebp
      ret

VESA_VWP_XLine        ENDP
```

Nah, sekarang yuk kita coba menggunakan fungsi-fungsi yang telah kita pelajari di atas dalam program.

Coba jalankan program LINE1.EXE. Terlihat empat buah garis. Dua garis horizontal yang dibentuk dengan fungsi VESA_HLine dan VESA_XHLine dan dua garis vertikal yang dibentuk dengan fungsi VESA_VLine dan VESA_XVLine. Garis yang terlihat transparantlah (berwarna putih di atas background hitam dan berwarna hitam di atas background putih) yang dibuat dengan menggunakan fungsi VESA_XHLine dan VESA_XVLine. Tampilan program ini seperti pada Gambar 5.11.



Gambar 5.11 Output dari Fungsi-fungsi Garis Lurus pada VGA.ASM

Background kotak-kotak hitam ini dibuat untuk membandingkan efek transparant antara dua macam fungsi garis.

PENJELASAN PROGRAM

```
/* ------------------------------------------------------------
Nama File : LINE1.CPP
Definisi  : Program mencoba Horizontal dan Vertical Line
Programmer: Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
```

Definisikan file header TRY.H.

```
char ch;
char * Buffer;

int main()
{
  int x,y;
  Buffer=(char *)malloc(MEM400);
  OpenVGA256(MOD400);
```

Buka mode graphic 640 x 400 256 warna dan sediakan memory untuk virtual screen pada variabel Buffer.

```
for (x=0;x<20;x++)
{
  for (y=0;y<20;y++)
  {
    if ((x+y)%2==1)
      VESA_Block(Buffer,x*32,y*20,x*32+31,y*20+19,7);
    else
      VESA_Block(Buffer,x*32,y*20,x*32+31,y*20+19,0);
  }
}
```

Fungsi di atas digunakan untuk menggambar kotak-kotak hitam putih di layar.

```
VESA_HLine(Buffer,50,50,150,15);
```

Panggil VESA_Hline dengan (X1,Y1) = (50,50) dan X2 = 150 dan warna = 15.

```
VESA_VLine(Buffer,250,50,350,15);
```

Panggil VESA_Vline dengan (X1,Y1) = (250,50) dan Y2 = 350 dan warna = 15.

```
VESA_XHLine(Buffer,50,110,150);
```

Panggil VESA_XHLine dengan (X1,Y1) = (50,110) dan X2 = 150.

```
VESA_XVLine(Buffer,335,50,350);
```

Panggil VESA_XVLine dengan (X1,Y1) = (335, 50) dan Y2 = 350.

```
VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen ke layar.

```
ch=getch();
```

Tunggu penekanan tombol keyboard.

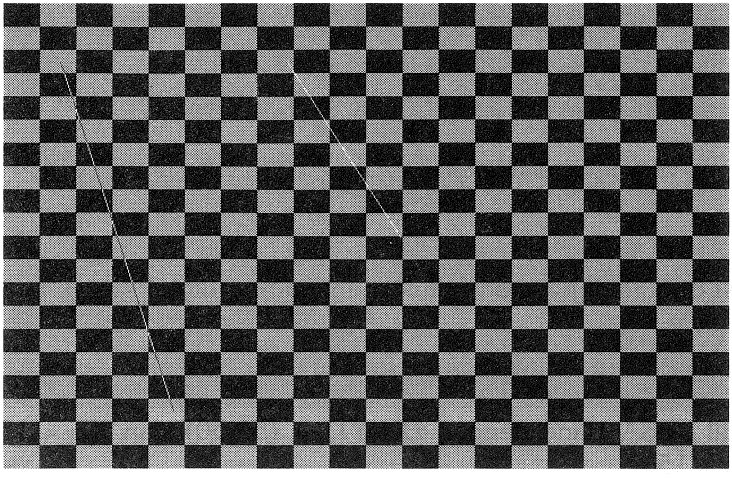
```
CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

Untuk mencoba mode fungsi garis bebas, coba jalankan program LINE2.EXE.

Output dari program ini dapat dilihat pada Gambar 5.12.



Gambar 5.12 Fungsi Garis Bebas dalam VGA.ASM

PENJELASAN PROGRAM

```
/*    ------------------------------------------------------
Nama File : LINE2.CPP
Definisi  : Program mencoba fungsi Garis Bebas.
Programmer: Agustinus Nalwan
------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"

char ch;
char * Buffer;

int main()
{
  int x,y;
  Buffer=(char *)malloc(MEM400);
  OpenVGA256(MOD400);
```

Buka mode graphic 640 x 400 256 warna dan ciptakan virtual screen pada variabel Buffer.

```
  for (x=0;x<20;x++)
  {
    for (y=0;y<20;y++)
    {
      if ((x+y)%2==1)
        VESA_Block(Buffer,x*32,y*20,x*32+31,y*20+19,7);
      else
        VESA_Block(Buffer,x*32,y*20,x*32+31,y*20+19,0);
    }
  }
```

Tampilkan background kotak-kotak hitam putih.

```
  VESA_VWP_XLine(Buffer,50,50,150,350);
```

Tampilkan garis bebas dengan mode transparant dari koordinat (50,50)-(150,350).

```
  VESA_Line(Buffer,250,50,350,200,15);
```

Tampilkan garis bebas dengan mode normal dari koordinat (250,50)-(350,200) dengan warna = 15.

```
  VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen ke layar.

```
  ch=getch();
```

Tunggu penekanan tombol keyboard.

```
  CloseVGA256();
```

Tutup mode graphic.

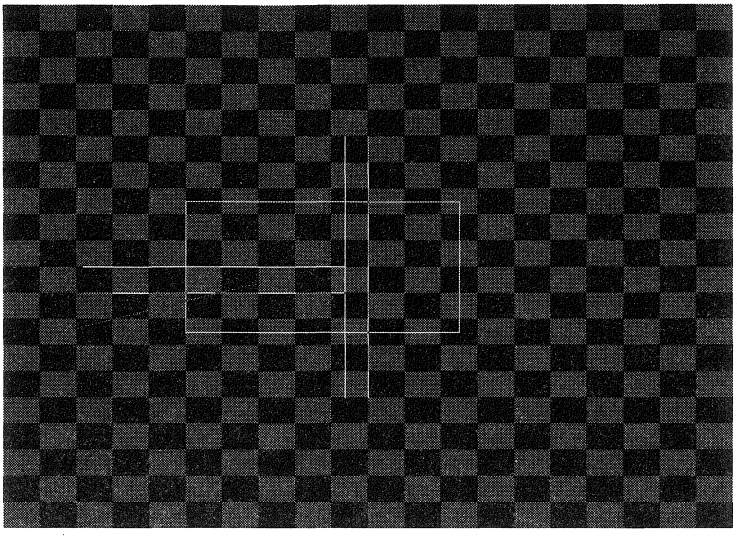
```
  return 1;
}
```

Fungsi VESA_VWP_Line

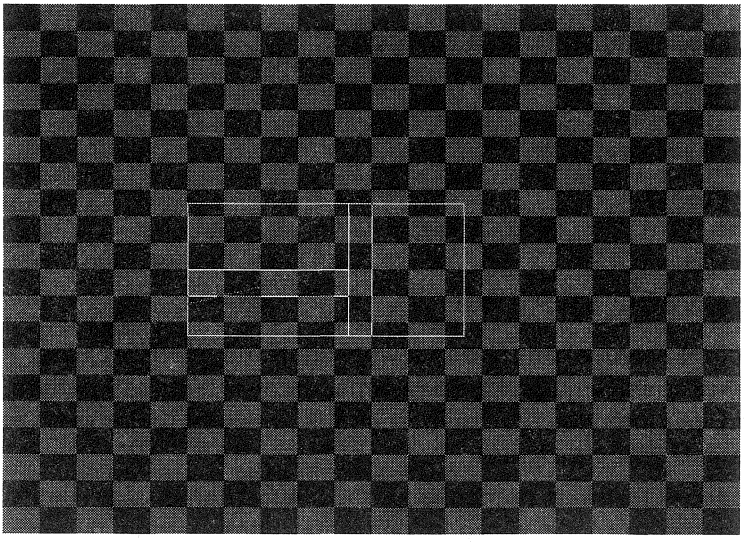
Fungsi ini merupakan mode viewport dari fungsi VESA_Line. Karena implementasinya juga hampir sama, hanya terdapat perbedaan di bagian pemotongan garisnya maka kita akan melompati pembahasan dari listing program.

Sekarang kita akan mencoba perbedaan antara fungsi-fungsi yang menggunakan mode viewport dan yang tidak menggunakan. Jalankan program VPSHAPE.EXE.

Akan ditampilkan beberapa garis lurus dan sebuah garis bebas di atas background kotak-kotak putih hitam. Garis kuning ialah viewport yang diciptakan. Karena fungsi-fungsi yang digunakan tidak mempertimbangkan viewport, maka garis-garis ini tidak terpotong oleh viewport tersebut. Lihat Gambar 5.13.



Gambar 5.13 Fungsi-fungsi Garis Mode Bukan Viewport



Gambar 5.14 Fungsi-fungsi Garis dengan Mode Viewport

Sekarang tekan tombol ENTER, penggambaran akan diulang dengan menggunakan mode viewport. Garis-garis terlihat dengan rapi terpotong oleh viewport. Lihat Gambar 5.14.

Fungsi-fungsi penggambaran bangun selanjutnya dirancang dalam file unit SHAPE.CPP dengan file header SHAPE.H.

Kenapa kok dirancang di luar file VGA.ASM? Karena fungsi-fungsi ini merupakan fungsi turunan yang artinya digunakannya fungsi-fungsi dalam VGA.ASM untuk merakit fungsi pada SHAPE.CPP ini.

Fungsi VESA_Rect

Fungsi ini digunakan untuk menampilkan persegi panjang pada virtual screen.

Parameter yang diterima ialah Buff sebagai alamat virtual screen, (X1,Y1)-(X2,Y2) sebagai koordinat persegi panjang. Col sebagai warna persegi panjang.

```
void VESA_Rect(char * Buff,long X1,long Y1,long X2,long
Y2,char col)
{
  VESA_VLine(Buff,X1,Y1,Y2,col);
```

Gambar sisi samping kiri persegi panjang.

```
  VESA_VLine(Buff,X2,Y1,Y2,col);
```

Gambar sisi samping kanan persegi panjang.

```
  VESA_HLine(Buff,X1,Y1,X2,col);
```

Gambar sisi atas persegi panjang.

```
  VESA_HLine(Buff,X1,Y2,X2,col);
```

Gambar sisi bawah persegi panjang.

```
}
```

Fungsi VESA_Xrect

Fungsi ini digunakan untuk menggambar persegi panjang dengan mode transparan, sehingga digunakan fungsi-fungsi garis lurus dengan mode transparan.

```
void VESA_XRect(char * Buff,long X1,long Y1,long X2,long
Y2)
{
```

```
  VESA_XVLine(Buff,X1,Y1,Y2);
  VESA_XVLine(Buff,X2,Y1,Y2);
  VESA_XHLine(Buff,X1,Y1,X2);
  VESA_XHLine(Buff,X1,Y2,X2);
}
```

Fungsi VESA_VWP_Rect

Fungsi ini digunakan untuk menggambar persegi panjang dengan mempertimbangkan viewport. Persegi panjang ini nantinya akan terpotong jika menerobos batas viewport. Sehingga digunakan fungsi-fungsi garis lurus yang menggunakan viewport.

```
void VESA_VWP_Rect(char * Buff,long X1,long Y1,long X2,long
Y2,char col)
{
  VESA_VWP_VLine(Buff,X1,Y1,Y2,col);
  VESA_VWP_VLine(Buff,X2,Y1,Y2,col);
  VESA_VWP_HLine(Buff,X1,Y1,X2,col);
  VESA_VWP_HLine(Buff,X1,Y2,X2,col);
}
```

Fungsi VESA_VWP_Xrect

Fungsi ini digunakan untuk menggambar persegi panjang dengan mempertimbangkan viewport dan dengan mode transparan.

```
void VESA_VWP_XRect(char * Buff,long X1,long Y1,long
X2,long Y2)
{
  VESA_VWP_XVLine(Buff,X1,Y1,Y2);
  VESA_VWP_XVLine(Buff,X2,Y1,Y2);
  VESA_VWP_XHLine(Buff,X1,Y1,X2);
  VESA_VWP_XHLine(Buff,X1,Y2,X2);
}
```

5.5 Fungsi-fungsi Penampilan Sprite

Selanjutnya kita akan membahas fungsi-fungsi yang menampilkan gambar sprite dalam library VGA.ASM. Fungsi penampilan gambar sprite dalam library VGA.ASM sangat lengkap. Disediakan fasilitas penampilan sprite transparan, penampilan sprite dengan ukuran bebas, penampilan sprite satu warna, penampilan cahaya, bayangan dan efek warna yang berbentuk sprite. Semua fungsi-fungsi penam-

pilan gambar yang disebutkan di atas membutuhkan adanya sebuah sprite yang akan ditampilkan. Sprite ini diciptakan dengan fungsi VESA_GetImage.

Catatan

Sprite ialah sebuah potongan gambar yang disimpan dalam memory. Setiap saat, potongan gambar ini dapat ditampilkan pada virtual screen. Sprite ini digunakan sebagai obyek-obyek dalam animasi seperti pemain, musuh, peluru, ledakan dan lain sebagainya.

Fungsi VESA_GetImage

Fungsi ini digunakan untuk merekam gambar dari virtual screen ke dalam suatu alamat memory. Potongan gambar yang ditangkap ini disebut sprite.

```
VESA_GetImage          PROC     NEAR

    push        ebp
    mov         ebp,esp
    ARG         PICBUFF:DWORD,X1:DWORD,Y1:DWORD,X2:DWORD,
                Y2:DWORD,BUFFER:DWORD
```

Parameter yang dikirimkan ialah koordinat kiri atas dan kanan bawah persegi panjang yang mengurung potongan gambar tersebut yaitu (X1,Y1) dan (X2,Y2). Juga PICBUFF yang berisi alamat memory virtual screen tempat gambar yang akan dipotong berada dan BUFFER yang berisi alamat memory di mana potongan gambar akan disimpan.

```
    mov         esi,[PICBUFF]
    mov         edi,[BUFFER]
```

Arahkan esi dan edi ke alamat virtual screen dan ke alamat tempat penyimpanan gambar.

```
    Mov         eax,[Y1]
    Mul         [A_SL]
    Add         eax,[X1]
```

Hitung jumlah byte antara titik kiri atas dari virtual screen ke titik pertama/kiri atas dari sprite yang akan diletakkan. Jarak ini selanjutnya akan disebut sebagai offset dari sprite. Caranya ialah dengan rumus:

```
Offset sprite  = Y * A_SL + X
```

di mana A_SL ialah jumlah titik horizontal.

```
Add          esi,eax
```

Tambahkan alamat memory tempat virtual screen berada dengan offset sprite sehingga kita mendapatkan alamat memory sesungguhnya dari kiri atas sprite.

```
Mov          ebx,[X2]
Sub          ebx,[X1]
Inc          ebx
```

Hitung panjang sprite dan masukkan ke dalam ebx.

```
Mov          [edi],bx
```

Masukkan panjang sprite ke dalam dua byte pertama pada alamat memory yang disediakan untuk menyimpan sprite.

```
Inc          edi
Inc          edi
Mov          ecx,[Y2]
Sub          ecx,[Y1]
Inc          ecx
```

Hitung lebar sprite dan masukkan ke dalam register ecx.

```
Mov          [edi],cx
```

Masukkan lebar sprite ke dalam dua byte berikutnya pada alamat memory yang disediakan untuk menyimpan sprite.

```
Inc          edi
Inc          edi
Cld

@vg1:
  Mov        edx,ecx
```

Kopikan lebar sprite ke register edx.

```
    Mov      ecx,ebx
```

Kopikan panjang sprite ke register ecx sebagai counter horizontal.

```
    Mov      eax,esi
```

Kopikan register esi ke dalam eax sehingga kita tidak akan kehilangan harga register esi setelah melakukan proses Mosvb berikutnya.

```
Rep         Movsb
```

Transfer data dari virtual screen ke alamat memory tempat sprite disimpan sejumlah panjang dari sprite.

```
Mov         esi,eax
```

Kembalikan harga esi ke harga awal sebelum dilakukannya Rep Movsb.

```
Add         esi,[A_SL]
```

Tambahkan esi dengan A_SL untuk mengakses koordinat Y selanjutnya dari gambar yang akan dipotong.

```
Mov         ecx,edx
```

Kembalikan lebar sprite ke registe ecx sebagai counter vertikal.

```
Loop @vg1
```

Loop hingga koordinat Y terakhir dari potongan gambar telah disimpan.

```
    mov         esp,ebp
    pop         ebp

    ret

VESA_GetImage         ENDP
```

Gambar yang dipotong oleh fungsi GetImage ini disimpan dalam suatu alamat memory yang dapat dipergunakan oleh fungsi-fungsi penampilan gambar yang akan dibahas berikutnya.

Fungsi VESA_PutImage

Fungsi ini ialah fungsi penampilan gambar yang paling sederhana.

```
VESA_PutImage         PROC        NEAR

        LOCAL   XWStart:DWord,XRStart:DWord,YWStart:DWord,
                YRStart:DWord,XRead:DWORD,Widths:DWord,
                YRead:DWord,Lengths:Dword
```

Variabel lokal dideklarasikan sebagai pembantu register untuk menyimpan harga.

```
push    ebp
mov     ebp,esp
sub     esp, 64
```

Lompati stack pointer dengan 64 byte untuk melompati alamat memory yang telah digunakan oleh variabel lokal. Sebenarnya fungsi AUTO_SIZE dapat digunakan sama seperti yang telah kita lakukan pada fungsi VESA_LINE yang secara otomatis akan menghitung jumlah memory yang digunakan untuk variabel lokal.

```
ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
        BUFFER:DWORD
```

Parameter yang dikirimkan ialah PICBUFFER yaitu alamat virtual screen yang akan digambari. X1 dan Y1 ialah koordinat di mana sprite akan ditampilkan. BUFFER ialah alamat memory tempat sprite yang akan ditampilkan disimpan.

```
mov     esi, [BUFFER]
mov     edi, [PICBUFFER]
```

Arahkan esi ke sprite dan edi ke virtual screen.

```
Xor     eax, eax
lodsw
mov     ecx, eax
Mov     [Lengths],eax
Mov     [XRead],eax
lodsw
Mov     [Widths],eax
Mov     [YRead],eax
Mov     [YRStart],0
```

Baca panjang dan lebar sprite. Kedua data ini disimpan dalam 4 byte pertama dari sprite. Masukkan ke dalam Xread dan Yread yang menunjukkan jumlah titik horizontal dan jumlah titik vertikal dalam sprite. Isi YRStart dengan 0. YRStart ini ialah nomor baris dimulainya penggambaran sprite. Variabel ini digunakan jika sprite dipotong terhadap batas atas layar sehingga baris pertama dari sprite yang akan digambar bukanlah baris nomor 0.

```
Mov     eax,[X1]
cmp     eax,S_L
jg      @PI_Vin
```

```
Add     eax,[Lengths]
Sub     eax,1
Cmp     eax,0
jl      @PI_Vin
Mov     eax,[Y1]
cmp     eax,S_W
jg      @PI_Vin
Add     eax,[Widths]
Sub     eax,1
Cmp     eax,0
jl      @PI_Vin
```

Deteksi, apakah sprite berada di luar layar. Jika ya, batalkan penggambaran sprite.

```
Mov     eax,[Y1]
Mov     [YWStart],eax
Mov     [XRStart],0
Mov     eax,[X1]
Mov     [XWStart],eax
```

Variabel XWStart dan YWStart menunjukkan koordinat X dan Y tempat sprite akan ditampilkan. Jika tidak terjadi pemotongan, koordinat ini akan sama persis dengan koordinat yang dikirimkan lewat parameter, tetapi jika terjadi pemotongan, koordinat ini akan dihitung kembali.

```
  cmp     eax,0
Jge @VESX1_Not_Negatives
  Neg     eax
  Mov     [XRStart],eax
  Mov     [XWStart],0
  Sub     [XRead],eax
```

Lakukan pemotongan sisi kiri layar. Jika terjadi pemotongan, kurangi jumlah titik horizontal (Xread) dengan jumlah titik yang dipotong. Juga set koordinat horizontal awal penggambaran yaitu XWStart menjadi batas kiri layar yaitu X = 0 dan set XRStart ke koordinat yang baru yang merupakan kolom pertama dari sprite yang akan ditampilkan.

```
@VESX1_Not_Negatives:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @VESX1_No_Right_Crops
```

```
Sub     eax,[S_L]
Sub     [XRead],eax
```

Lakukan pemotongan sisi kanan layar.

```
@VESX1_No_Right_Crops:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @VESNo_Top_Crops
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
```

Lakukan pemotongan sisi atas layar.

```
@VESNo_Top_Crops:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @VESNo_Bot_Crops
  Sub     eax,[S_W]
  Sub     [YRead],eax
```

Lakukan pemotongan sisi bawah layar.

```
@VESNo_Bot_Crops:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax
```

Hitung offset memory dari titik pertama sprite.

```
Mov     eax,[YRStart]
Mul     [Lengths]
Add     esi,eax
Add     esi,[XRStart]
```

Hitung jumlah byte yang dilompati antara titik pertama dari sprite, sebelum dipotong ke titik pertama dari sprite setelah dipotong. Jumlah byte ini nantinya akan kita namakan sebagai offset sprite untuk mempermudah pembahasan.

```
Mov     ecx, [YRead]
```

Sediakan counter vertikal.

```
@VESvLoop1:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
```

Sediakan counter horizontal.

```
@VESvLoop2:
   Lodsb
   or al,al
   Jz @VESvNo_Write
   Stosb
```

Ambil warna titik pada sprite dan deteksi apakah warna 0. Jika tidak, lakukan peletakkan titik pada virtual screen.

```
     dec edi
 @VESvNo_Write:
   inc edi
```

Tambah edi untuk memproses koordinat horizontal berikutnya.

```
Loop @VESvLoop2
```

Loop hingga seluruh titik horizontal selesai diletakkan.

```
Pop esi
Pop edi
Add esi,[Lengths]
Add edi,[A_SL]
```

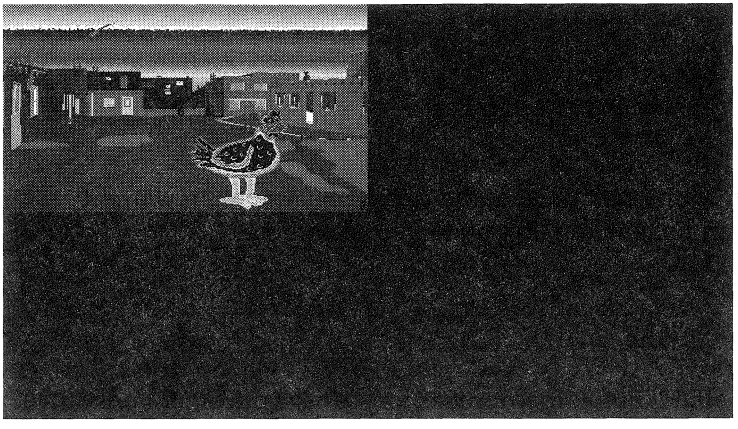
Tambahkan esi dan edi untuk memproses koordinat vertikal berikutnya.

```
Pop ecx
Loop @VESvLoop1
```

Loop hingga seluruh sprite telah diletakkan.

```
@PI_Vin:
      mov esp,ebp
      pop ebp

      ret

VESA_PutImage                 ENDP
```

Untuk mencoba fungsi di atas, coba jalankan program IMAGE1.EXE. Mula-mula akan ditampilkan gambar kota. Tekan ENTER, akan ditampilkan gambar ayam. Tekan ENTER sekali lagi, sekarang akan ditampilkan gambar ayam di atas kota. Coba gerakkan kursor untuk menggerakkan ayam. Output dari program ini dapat dilihat pada Gambar 5.15.



Gambar 5.15 Sprite yang Diletakkan di atas Background

Gambar ayam ini ialah sprite yang ditampilkan dengan menggunakan fungsi VESA_PutImage.

Memang warna tampilan program terlihat sedikit kacau. Hal ini karena kita belum menggunakannya sistem palette yang berada pada unit VGAPAL.CPP. Unit ini akan kita bahas berikutnya.

PENJELASAN PROGRAM

```
/* ------------------------------------------------------
Nama File : IMAGE1.CPP
Definisi  : Program mencoba menggerakkan sprite
Programmer: Agustinus Nalwan
------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"

char ch;
```

```
char * Buffer;
char * Picture;
char * Back;
```

Deklarasikan virtual screen Buffer dan Back. Deklarasikan variabel pointer tempat untuk menyimpan sprite yaitu pada variabel Picture.

```
void LoadPicture(char * fname, char * Buff)
{
  FILE * fpic;
  int x,y;
  char col;
  fpic=fopen(fname,"rb");
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
      fread(&col,1,1,fpic);
      VESA_PutPixel(Buff,x,y,col);
    }
  }
  fclose(fpic);
```

Buka file gambar dan letakkan titik satu per satu pada virtual screen Buff.

```
}

int main()
{
  int finish=0,x=160,y=100;
```

Inisialisasi koordinat (X,Y) pada (160,100).

```
  Buffer=(char *)malloc(MEM400);
  Back=(char *)malloc(MEM400);
```

Alokasikan memory untuk virtual screen Buffer dan Back.

```
  Picture=(char *)malloc(15000);
```

Alokasikan memory untuk menyimpan sprite. Adapun cara menghitung jumlah memory untuk sprite ini ialah dengan rumus:

Jumlah memory = 4 + Panjang x Lebar x Byte Setiap titik

Karena sprite yang akan kita ambil ini berukuran 92 x 99 dan karena kita menggunakan mode graphic 8 bit (8 bit = 1 byte setiap titik)

maka jumlah memory yang diperlukan ialah 4 + 92 x 99 x 1 = sekitar 10000 Byte. Untuk amannya kita ambil 15000 Byte.

```
OpenVGA256(MOD400);
VESA_ClearDouble(Back,0);
```

Buka mode graphic 640 x 400 256 warna dan hapus virtual screen Back dengan warna hitam.

```
LoadPicture("town.pic",Back);
VESA_MoveToScreen(Back);
ch=getch();
```

Baca file gambar TOWN.PIC dan masukkan ke dalam virtual screen Back. Kemudian tampilkan virtual screen Back ini pada layar monitor.

Tunggu penekanan tombol keyboard.

```
VESA_ClearDouble(Buffer,0);
LoadPicture("chicken.pic",Buffer);
```

Hapus virtual screen Buffer dengan warna hitam. Baca file CHICKEN.PIC ke dalam virtual screen Buffer.

```
VESA_GetImage(Buffer,2,52,93,150,Picture);
```

Ambil sprite ayam di koordinat (2,52)-(93,150) dalam virtual screen Buffer. Masukkan sprite ini dalam variabel Picture.

```
VESA_MoveToScreen(Buffer);
ch=getch();
```

Tampilkan Virtual screen ini ke layar monitor dan tunggu penekanan tombol keyboard.

```
VESA_ClearDouble(Buffer,0);
while (!finish)
{
  VESA_MoveDouble(Back,Buffer);
  VESA_PutImage(Buffer,x,y,Picture);
  VESA_MoveToScreen(Buffer);
```

Pindahkan gambar kota dari virtual screen Back ke virtual screen Buffer, kemudian tampilkan sprite pada virtual screen Buffer di atas gambar kota dan tampilkan virtual screen Buffer ke layar monitor.

```
  while (!kbhit())
  {
```

```
    }
    ch=getch();
    if (ch==72) y--;
    if (ch==80) y++;
    if (ch==75) x--;
    if (ch==77) x++;
```

Gerakkan koordinat sprite sesuai dengan penekanan tombol kursor.

```
    if (ch==27) finish=1;
```

Keluar jika tombol ESC ditekan.

```
  }
  CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

Fungsi VESA_LPutImage

Jika kita hendak menampilkan gambar panah yang sedang berjalan ke kanan, kita cukup menyediakan sprite bergambar panah yang sedang menghadap ke kanan kemudian kita cukup memindahkan koordinat pada setiap loop sehingga panah seolah-olah bergerak ke kanan. Tetapi jika kita hendak menampilkan beberapa panah yang beberapa di antaranya menghadap ke kanan dan beberapa di antaranya menghadap ke kiri, perlukah kita menciptakan dua buah sprite di mana yang satu menghadap ke kiri dan yang satu menghadap ke kanan? Pada seri-seri sebelumnya, kita selalu menggunakan dua buah gambar untuk situasi ini, sekarang tidak lagi. Fungsi VESA_LPutImage ialah fungsi pencerminan horizontal dari fungsi VESA_PutImage.

```
VESA_LPutImage          PROC        NEAR

        LOCAL   XWStart:DWord,XRStart:DWord,YWStart:DWord,
                YRStart:DWord,XRead:DWORD,Widths:DWord,
                YRead:DWord,Lengths:DWord

        push    ebp
        mov     ebp,esp
        sub     esp, 64
        ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                BUFFER:DWORD
```

```
mov     esi, [BUFFER]
mov     edi, [PICBUFFER]

Xor     eax, eax
lodsw
mov     ecx, eax
Mov     [Lengths],eax
Mov     [XRead],eax
lodsw
Mov     [Widths],eax
Mov     [YRead],eax
Mov     [YRStart],0
```

Baca panjang dan lebar sprite. Isikan Yread dan YRStart.

```
Mov     eax,[X1]
cmp     eax,S_L
jg      @MrPI_Vin
Add     eax,[Lengths]
Sub     eax,1
Cmp     eax,0
jl      @MrPI_Vin
Mov     eax,[Y1]
cmp     eax,S_W
jg      @MrPI_Vin
Add     eax,[Widths]
Sub     eax,1
Cmp     eax,0
jl      @MrPI_Vin
```

Batalkan penggambaran sprite jika sprite berada di luar layar.

```
Mov     eax,[Y1]
Mov     [YWStart],eax
Mov     [XRStart],0
Mov     eax,[X1]
Mov     [XWStart],eax
cmp     eax,0
Jge @MrVESX1_Not_Negatives
Neg     eax
Mov     [XRStart],eax
Mov     [XWStart],0
Sub     [XRead],eax
```

Lakukan pemotongan kiri layar.

```
@MrVESX1_Not_Negatives:
Mov     eax,[X1]
```

```
    Add     eax,[Lengths]
    Dec     eax
    Cmp     eax,[S_L]
    Jle     @MrVESX1_No_Right_Crops
    Sub     eax,[S_L]
    Sub     [XRead],eax
```

Lakukan pemotongan kanan layar.

```
@MrVESX1_No_Right_Crops:
    Mov     eax,[Y1]
    Cmp     eax,0
    Jge     @MrVESNo_Top_Crops
    Neg     eax
    Mov     [YWStart],0
    Mov     [YRStart],eax
    Sub     [YRead],eax
```

Lakukan pemotongan atas layar.

```
@MrVESNo_Top_Crops:
    Mov     eax,[Y1]
    Add     eax,[Widths]
    Dec     eax
    Cmp     eax,[S_W]
    Jle     @MrVESNo_Bot_Crops
    Sub     eax,[S_W]
    Sub     [YRead],eax
```

Lakukan pemotongan bawah layar.

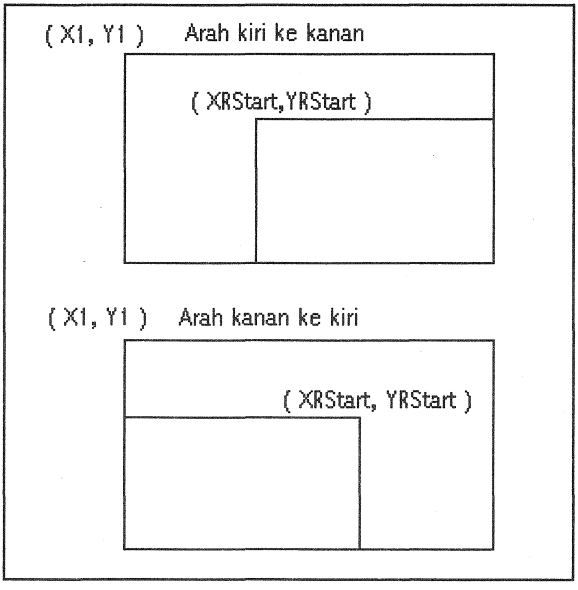
```
@MrVESNo_Bot_Crops:
    Add     edi,[XWStart]
    Mov     eax,[YWStart]
    mul     [A_SL]
    Add     edi,eax
```

Hitung offset memory titik pertama yang akan digambar.

```
    Mov     eax,[YRStart]
    Mul     [Lengths]
    Add     esi,eax
    add     esi,[Lengths]
    dec     esi
    sub     esi,[XRStart]
```

Hitung offset sprite dari titik pertama pada sprite yang akan dibaca. Proses perhitungan di sini agak lain. Pada VESA_PutImage, perhi-

tungan offset sprite dilakukan dengan cara YRStart x Lengths + XRStart karena titik-titik pada sprite dibaca dari kiri ke kanan. Di sini, dilakukan pembacaan titik-titik dari kanan ke kiri sehingga gambar yang dihasilkan dicerminkan horizontal. Sehingga perhitungan offset sprite menjadi Y x Lenghts + Lengths-XRStart. Perhatikan Gambar 5.16. Di sini kotak kecil menunjukkan sprite yang akan ditampilkan setelah dipotong oleh batas atas layar dan batas kiri layar. Gambar pertama menunjukkan penggunaan VESA_PutImage dengan arah normal untuk pembacaan titik dalam sprite. Gambar kedua menunjukkan penggunaan VESA_LPutImage dengan arah kebalikan untuk pembacaan titik dalam sprite. Sehingga bagian sprite yang terpotong ialah bagian kanan walaupun pemotongan yang dilakukan ialah pemo- tongan batas kiri layar. Hal ini karena sisi kanan sprite sebenarnya akan ditampilkan sebagai sisi kiri (pencerminan horizontal).



Gambar 5.16 Proses Perhitungan Offset Sprite Antara VESA_Putimage dan VESA_Lputimage

```
  Mov     ecx, [YRead]
@MrVESvLoop1:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
```

```
@MrVESvLoop2:
   mov     al,[esi]
   dec     esi
```

Kalau dulunya kita gunakan penambahan esi, sekarang kita lakukan pengurangan esi untuk mengakses koordinat horizontal sebelumnya. Karena arah pembacaan titik dari kanan ke kiri.

```
      or al,al
      Jz @MrVESvNo_Write
      Stosb
      dec edi
   @MrVESvNo_Write:
      inc edi
   Loop @MrVESvLoop2
   Pop esi
   Pop edi
   Add esi,[Lengths]
   Add edi,[A_SL]
   Pop ecx
   Loop @MrVESvLoop1

@MrPI_Vin:
   mov esp,ebp
   pop ebp
   ret

VESA_LPutImage                  ENDP
```

Fungsi VESA_Put1Col

Fungsi ini digunakan untuk menampilkan sprite dengan menggunakan sebuah warna. Sprite ini hanya dimanfaatkan bentuknya saja.

Implementasi fungsi ini sama persis seperti VESA_PutImage. Hanya terdapat perbedaan pada bagian pengisian titik.

```
VESA_Put1Col        PROC        NEAR

      LOCAL    XWStart:DWord,XRStart:DWord,YWStart:DWord,
               YRStart:DWord,XRead:DWORD,Widths:DWord,
               YRead:DWord,Lengths:DWord=AUTO_SIZE

      push     ebp
      mov      ebp,esp
      sub      esp, AUTO_SIZE
      ARG      PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
               BUFFER:DWORD,COL:BYTE
```

Parameter yang dikirimkan ialah PICBUFFER yaitu alamat virtual screen yang akan dikirimkan. (X1,Y1) adalah koordinat sprite yang akan diletakkan, sedangkan BUFFER ialah alamat sprite. COL ialah warna yang akan diletakkan pada virtual screen.

```
mov     esi, [BUFFER]
mov     edi, [PICBUFFER]
```

Arahkan esi dan edi ke BUFFER dan PICBUFFER.

```
Xor     eax, eax
lodsw
mov     ecx, eax
Mov     [Lengths],eax
Mov     [XRead],eax
lodsw
Mov     [Widths],eax
Mov     [YRead],eax
Mov     [YRStart],0
```

Baca panjang dan lebar sprite dan inisialisasikan Xread, Yread dan YRStart.

```
Mov     eax,[X1]
cmp     eax,S_L
jg      @PI1_Vin
Add     eax,[Lengths]
Sub     eax,1
Cmp     eax,0
jl      @PI1_Vin
Mov     eax,[Y1]
cmp     eax,S_W
jg      @PI1_Vin
Add     eax,[Widths]
Sub     eax,1
Cmp     eax,0
jl      @PI1_Vin
```

Batalkan proses penggambaran sprite jika sprite berada di luar layar.

```
Mov     eax,[Y1]
Mov     [YWStart],eax
Mov     [XRStart],0
Mov     eax,[X1]
Mov     [XWStart],eax
cmp     eax,0
Jge @VESX1_Not_Negativescol
Neg     eax
```

```
  Mov     [XRStart],eax
  Mov     [XWStart],0
  Sub     [XRead],eax
```

Lakukan pembatasan kiri layar.

```
@VESX1_Not_Negativescol:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @VESX1_No_Right_Cropscol
  Sub     eax,[S_L]
  Sub     [XRead],eax
```

Lakukan pembatasan kanan layar.

```
@VESX1_No_Right_Cropscol:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @VESNo_Top_Cropscol
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
```

Lakukan pembatasan atas layar.

```
@VESNo_Top_Cropscol:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @VESNo_Bot_Cropscol
  Sub     eax,[S_W]
  Sub     [YRead],eax
```

Lakukan pembatasan bawah layar.

```
@VESNo_Bot_Cropscol:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax
```

Hitung offset memory titik pertama pada virtual screen di mana sprite akan diletakkan.

```
  Mov     eax,[YRStart]
```

```
        Mul     [Lengths]
        Add     esi,eax
        Add     esi,[XRStart]
```

Hitung offset sprite dari titik pertama sprite yang akan ditampilkan.

```
        Mov     ecx, [YRead]
```

Sediakan counter vertikal.

```
        Mov     dl,[Col]
      @VESvLoop1col:
        Push ecx
        Push edi
        Push esi
        Mov ecx, [XRead]
```

Sediakan counter horizontal.

```
      @VESvLoop2col:
         Lodsb
         or al,al
         Jz @VESvNo_Writecol
```

Jika warna 0 yang dibaca maka jangan tampilkan titik.

```
         mov al,dl
         Stosb
```

Tampilkan dl pada layar di mana dl menyimpan warna (Col). Sehingga warna sebenarnya dari sprite tidak dipergunakan sebagai warna titik yang diletakkan.

```
           dec edi
       @VESvNo_Writecol:
           inc edi
      Loop @VESvLoop2col
      Pop esi
      Pop edi
      Add esi,[Lengths]
      Add edi,[A_SL]
      Pop ecx
      Loop @VESvLoop1col
@PI1_Vin:
      mov esp,ebp
      pop ebp
      ret

VESA_Put1Col                    ENDP
```

Fungsi VESA_LPut1Col

Fungsi ini ialah mode pencerminan horizontal dari fungsi VESA_Put1Col. Implementasi fungsi ini hampir sama dengan fungsi VESA_LPutImage yaitu dilakukannya perhitungan offset sprite yang berbeda.

```
VESA_LPut1Col          PROC        NEAR

        LOCAL    XWStart:DWord,XRStart:DWord,YWStart:DWord,
                 YRStart:DWord,XRead:DWORD,Widths:DWord,
                 YRead:DWord,Lengths:DWord=AUTO_SIZE

        push     ebp
        mov      ebp,esp
        sub      esp, AUTO_SIZE
        ARG      PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                 BUFFER:DWORD,COL:BYTE
```

Parameter yang dikirimkan ialah PICBUFFER yaitu alamat virtual screen yang akan ditampilkan. (X1,Y1) ialah koordinat sprite yang akan ditampilkan dan BUFFER yaitu sprite yang akan ditampilkan. COL ialah warna titik yang akan diletakkan sebagai pengganti warna sprite.

```
        mov      esi, [BUFFER]
        mov      edi, [PICBUFFER]
```

Arahkan esi dan edi ke BUFFER dan PICBUFFER.

```
        Xor      eax, eax
        lodsw
        mov      ecx, eax
        Mov      [Lengths],eax
        Mov      [XRead],eax
        lodsw
        Mov      [Widths],eax
        Mov      [YRead],eax
        Mov      [YRStart],0
```

Baca panjang dan lebar sprite dan inisialisasikan XRead, YRead, YR-Start.

```
        Mov      eax,[X1]
        cmp      eax,S_L
        jg       @MrPI1_Vin
        Add      eax,[Lengths]
        Sub      eax,1
```

```
  Cmp     eax,0
  jl      @MrPI1_Vin
  Mov     eax,[Y1]
  cmp     eax,S_W
  jg      @MrPI1_Vin
  Add     eax,[Widths]
  Sub     eax,1
  Cmp     eax,0
  jl      @MrPI1_Vin
```

Batalkan penggambaran sprite jika sprite berada di luar layar.

```
  Mov     eax,[Y1]
  Mov     [YWStart],eax
  Mov     [XRStart],0
  Mov     eax,[X1]
  Mov     [XWStart],eax
  cmp     eax,0
Jge @MrVESX1_Not_Negativescol
  Neg     eax
  Mov     [XRStart],eax
  Mov     [XWStart],0
  Sub     [XRead],eax
```

Potong sprite terhadap kiri layar.

```
@MrVESX1_Not_Negativescol:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @MrVESX1_No_Right_Cropscol
  Sub     eax,[S_L]
  Sub     [XRead],eax
```

Potong sprite terhadap kanan layar.

```
@MrVESX1_No_Right_Cropscol:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @MrVESNo_Top_Cropscol
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
```

Potong sprite terhadap atas layar.

```
@MrVESNo_Top_Cropscol:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @MrVESNo_Bot_Cropscol
  Sub     eax,[S_W]
  Sub     [YRead],eax
```

Potong sprite terhadap bawah layar.

```
@MrVESNo_Bot_Cropscol:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax
```

Hitung offset memory titik pertama sprite pada virtual screen yang akan ditampilkan.

```
  Mov     eax,[YRStart]
  Mul     [Lengths]
  Add     esi,eax
  Add     esi,[Lengths]
  dec     esi
  Sub     esi,[XRStart]
```

Hitung offset sprite dari titik pertama sprite yang akan ditampilkan.

```
  Mov     ecx, [YRead]
  Mov     dl,[Col]
@MrVESvLoop1col:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
@MrVESvLoop2col:
   mov al,[esi]
   dec esi
```

Baca titik pada sprite dan kurangi esi untuk mengakses koordinat horizontal sebelumnya.

```
   or al,al
   Jz @MrVESvNo_Writecol
   mov al,dl
   Stosb
   dec edi
```

```
        @MrVESvNo_Writecol:
          inc edi
       Loop @MrVESvLoop2col
       Pop esi
       Pop edi
       Add esi,[Lengths]
       Add edi,[A_SL]
       Pop ecx
       Loop @MrVESvLoop1col
@MrPI1_Vin:
       mov esp,ebp
       pop ebp
       ret

VESA_LPut1Col                    ENDP
```

Fungsi VESA_Put3D

Fungsi ini digunakan untuk meletakkan sprite dengan ukuran bebas. Maksud ukuran bebas di sini ialah sprite yang berukuran 30 x 80 dapat kita tampilkan dengan ukuran 140 x 20 ataupun 20 x 100 dan lain sebagainya. Kita dapat meminta berapa ukuran sprite ini nantinya pada saat ditampilkan.

Implementasi fungsi ini sama persis seperti pada fungsi Put3D dalam VGACAD.TPU. Hanya saja di sini digunakan 32 bit programming.

Tekniknya sebenarnya ialah dengan mengulang beberapa titik atau dengan tidak menampilkan beberapa titik di dalam sprite, sehingga diperoleh ukuran yang diinginkan.

Implementasi teknik ini ialah dengan cara menggunakan bilangan pecahan sebagai counter dari pembacaan titik sprite. Misalnya sprite kita berukuran 50 x 50 sedangkan kita hendak menampilkannya dengan ukuran 100 x 100. Kita akan mengadakan penulisan 100 titik horizontal sehingga nantinya kita akan mengadakan loop horizontal sejumlah 100 kali. Karena ukuran horizontal sprite ialah 50 maka kita harus membaca titik-titik dalam sprite dengan lompatan horizontal lebih kecil dari 1 yaitu 50/100 = 0.5 (lompatan horizontal ini dinamakan RX) sehingga 50 titik ini dibagi secara merata dalam 100 loop. Demikian pula dengan lompatan vertikal (RY). Karena kita membutuhkan bilangan pecahan, maka kita akan mempergunakan metode fixed point.

```
VESA_Put3D    PROC NEAR

        LOCAL    XWStart:DWord,YWStart:DWord,Length:DWord,
```

```
        Width:DWord,XRead:DWord,YRead:DWord,XFixed:
        DWord,YFixed:DWord,XFStart:DWord,YFStart:DWord,
        RX:DWord,RY:DWord

Push    ebp
mov     ebp,esp
sub     esp, 96
ARG     PICBUFF:DWORD,X:DWORD,Y:DWORD,XL:DWORD,
        YL:DWORD,BUFFER:DWORD
```

Parameter yang dikirimkan ialah PICBUFF yaitu alamat virtual screen yang akan digambari. X dan Y yaitu koordinat di mana sprite akan ditampilkan. XL dan YL yaitu ukuran sprite yang akan ditampilkan dan BUFFER yang berisi alamat memory tempat sprite berada.

```
mov     esi,[BUFFER]
mov     edi,[PICBUFF]

xor     eax,eax
Lodsw
Mov     [Length],eax

shl     eax,16
Mov     ebx,eax
Xor     eDx,eDx
Div     [XL]
```

Mula-mula kita harus menghitung Rx dan Ry dalam format fixed point. Rx diperoleh dengan membagi Length dengan XL.

```
Mov     [Rx],eax
Mov     eax,[XL]
Mov     [XRead],eax

xor     eax,eax
Lodsw
Mov     [Width],eAx
shl     eax,16
Mov     ebx,eax
Xor     eDx,eDx
Div     [YL]
Mov     [Ry],eAx
```

Ry diperoleh dengan membagi Width dengan YL.

```
Mov     eAx,eBx
Xor     eDx,eDx
Mov     eax,[YL]
```

```
    Mov     [YRead],eAx
    Mov     [XFixed],0
    Mov     [YFixed],0
    Mov     [XFStart],0
    Mov     [YFStart],0
    Mov     eAx,[Y]
    Mov     [YWStart],eAx
```

Inisialisasikan Xfixed, Yfixed, XFStart dan YFStart yang semuanya dalam format fixed point.

```
    Mov     eAx,[X]
    Mov     [XWStart],eAx
    Cmp eAx,0
Jge @VESP3_X1_Not_Negative
    Neg eAx
    Mov XWStart,0
    Sub XRead,eAx
    Mul Rx
    Mov XFStart,eAx
```

Lakukan pembatasan kiri layar. Jika ternyata sprite memotong kiri layar, isi XWStart (koordinat kiri atas diletakkannya sprite) dengan koordinat X = 0. Kemudian kurangi XRead dengan jumlah titik yang terpotong. Isi XFStart yang merupakan koordinat titik pertama pada virtual screen di mana sprite akan diletakkan dengan jumlah titik yang terpotong x Rx karena setiap titik memiliki ukuran horizontal sebesar Rx titik.

```
@VESP3_X1_Not_Negative:
    Mov eAx,XWStart
    Add eAx,XRead
    Dec eAx
    Cmp eAx,[S_L]
```

Deteksi perpotongan sprite dengan kanan layar.

```
Jle @VESP3_X1_No_Right_Crop
    Sub eAx,[S_L]
    Sub XRead,eAx
@VESP3_X1_No_Right_Crop:
    Mov eAx,Y
    Cmp eAx,0
Jge @VESP3_No_Top_Crop
    Neg eAx
    Mov YWStart,0
    Sub YRead,eAx
```

```
    Mul Ry
    Mov YFStart,eAx
```

Deteksi perpotongan sprite dengan atas layar.

```
@VESP3_No_Top_Crop:
    Mov eAx,YWStart
    Add eAx,YRead
    Cmp eAx,[S_W]
Jle @VESP3_No_Bot_Crop
    Sub eAx,[S_W]
    Sub YRead,eAx
```

Deteksi perpotongan sprite dengan bawah layar.

```
@VESP3_No_Bot_Crop:
    Add edi,[XWStart]
    Mov eAx,[YWStart]
    mul [A_SL]
    Add edi,eAx
```

Hitung offset memory titik pertama sprite pada virtual screen.

```
    Mov eCx,[YRead]
    Mov eAx,[YFStart]
    Mov [YFixed],eAx
```

Sediakan counter sebesar YRead untuk counter jumlah titik vertikal yang harus diletakkan.

```
@VESP3_Loop1:
  Push eCx
  Push edi
  Push esi
  Mov eAx,[YFixed]
  shr eax,16
  Mul [Length]
  Add esi,eAx
```

Hitung offset titik sprite paling kiri dari koordinat Y yang sedang diakses.

```
  Mov eCx,[XRead]
  Mov eAx,[XFStart]
  Mov [XFixed],eAx
  xor ebx,ebx
  Mov eBx,[Rx]
```

Sediakan counter sepanjang jumlah titik horizontal yang harus diletakkan. Isi variabel Xfixed dengan XFStart yaitu koordinat kiri sprite dalam format fixed point untuk siap-siap melakukan pengisian titik dari kiri ke kanan.

```
@VESP3_Loop2:
   Mov eAx,[XFixed]
   shr eax,16
   Push esi
   Add esi,eAx
```

Hitung offset sprite dalam format desimal.

```
   Lodsb
   Pop esi
   Add [XFixed],eBx
```

Baca titik kemudian tambahkan koordinat X dari titik yang sedang diproses dengan RX untuk mengakses titik horizontal selanjutnya dalam sprite.

```
   or al,al
   Jz @VESP3_No_Write
   Stosb
```

Jika warna titik = 0 maka jangan tampilkan titik.

```
   dec edi
@VESP3_No_Write:
   inc edi
```

Proses koordinat X selanjutnya.

```
Loop @VESP3_Loop2
```

Loop hingga seluruh titik horizontal telah digambar.

```
Pop esi
Pop edi
xor ebx,ebx
Mov eBx,[Ry]
Add [YFixed],eBx
Add edi,[A_SL]
```

Tambahkan koordinat Y dengan RY untuk mengakses koordinat Y berikutnya dari sprite.

```
Pop eCx
Loop @VESP3_Loop1
```

Loop hingga seluruh sprite telah digambar.

```
        mov   esp,ebp
        pop   ebp

        ret
VESA_Put3D    ENDP
```

Catatan

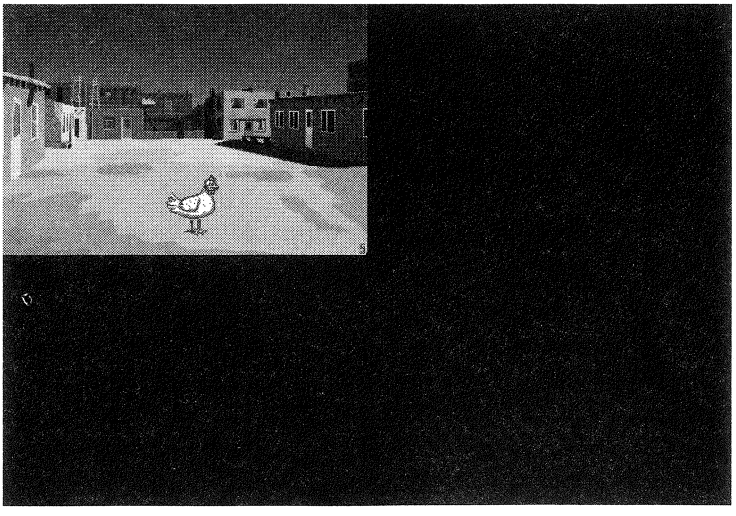
Pembahasan secara detil dari VESA_Put3D ini dapat diikuti dalam seri Pemrograman Animasi dan Game Profesional 2.

Nah, sekarang cobalah program PUT3D.EXE.

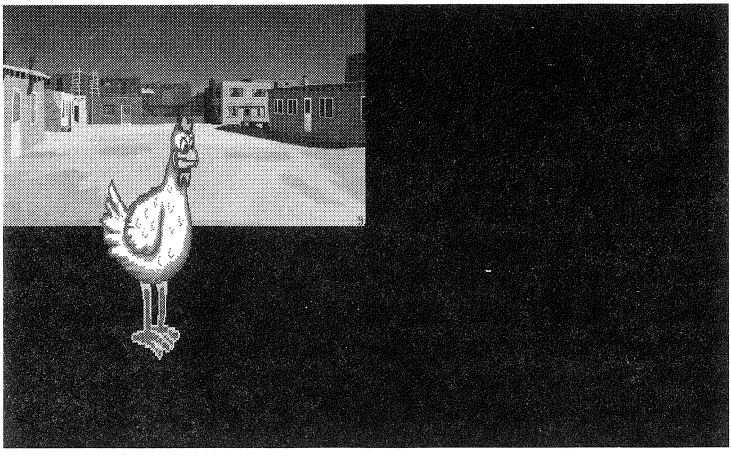
Tekan ENTER dua kali untuk melompati proses pembacaan background dan proses perekaman sprite ayam.

Nah, sekarang coba tekan tombol kursor untuk menggerakkan ayam. Tekan tombol Q/A untuk memperbesar/memperkecil ukuran ayam secara vertikal, dan W/S untuk memperbesar/memperkecil ukuran ayam secara horizontal. Tekan ESC untuk keluar.

Gambar 5.17 menunjukkan sprite dengan ukuran kecil dan pada Gambar 5.18 sprite telah diubah menjadi lebih besar.



Gambar 5.17 Output Fungsi Penampilan Sprite Berukuran Bebas



Gambar 5.18 Output Fungsi Penampilan Sprite Berukuran Bebas

PENJELASAN PROGRAM

```
/*     ------------------------------------------------------
Nama File : PUT3D.CPP
Definisi  : Program mencoba peletakan sprite tiga dimensi
Programmer: Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
```

Definisikan library VGAPAL.CPP beserta file headernya VGAPAL.H. Library ini digunakan untuk mengatur susunan warna palette dalam tabel warna seperti yang telah dijelaskan pada awal bab ini.

```
char ch;
char * Buffer;
char * Picture;
char * Back;

void LoadPicture(char * fname, char * Buff)
{
  FILE * fpic;
```

```
  int x,y;
  char col;
  fpic=fopen(fname,"rb");
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
      fread(&col,1,1,fpic);
      VESA_PutPixel(Buff,x,y,col);
    }
  }
  fclose(fpic);
}
```

Tampilkan gambar kota dari file ke virtual screen.

```
int main()
{
  int finish=0,x=160,y=100;
```

Letakkan koordinat awal sprite = (160,100).

```
  int sx=50,sy=50;
```

Set ukuran dari sprite sebesar (50,50).

```
  Buffer=(char *)malloc(MEM400);
  Back=(char *)malloc(MEM400);
```

Alokasikan memory untuk virtual screen Buffer dan Back.

```
  Picture=(char *)malloc(15000);
```

Alokasikan memory untuk sprite sebesar 15000 Byte.

```
  OpenVGA256(MOD400);
```

Buka mode graphic.

```
  Install_Palette("dev.pal");
  Refresh_Palette();
```

Install palette dari file DEV.PAL. Kemudian aktifkan susunan palette ke layar monitor.

```
  VESA_ClearDouble(Back,0);
```

Hapus virtual screen Back dengan warna 0.

```
  LoadPicture("town.pic",Back);
```

Baca file gambar TOWN.PIC ke virtual screen Back.

```
VESA_MoveToScreen(Back);
ch=getch();
```

Tampilkan virtual screen Back ke layar monitor dan tunggu penekanan tombol keyboard.

```
VESA_ClearDouble(Buffer,0);
LoadPicture("chicken.pic",Buffer);
VESA_GetImage(Buffer,2,52,93,150,Picture);
```

Hapus virtual screen Buffer dengan warna 0 kemudian tampilkan gambar CHICKEN.PIC ke virtual screen Buffer. Ambil sprite ayam ke alamat memory Picture.

```
VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen Buffer ke layar monitor.

```
ch=getch();
VESA_ClearDouble(Buffer,0);
while (!finish)
{
  VESA_MoveDouble(Back,Buffer);
  VESA_Put3D(Buffer,x,y,sx,sy,Picture);
```

Pindahkan gambar kota dari virtual screen Back ke virtual screen Buffer.

Tampilkan gambar sprite ke virtual screen Buffer di atas gambar kota.

```
  VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen Buffer ke layar monitor.

```
  while (!kbhit())
  {
  }
  ch=getch();
  if (ch==72) y--;
  if (ch==80) y++;
  if (ch==75) x--;
  if (ch==77) x++;
```

Gerakkan sprite sesuai dengan gerakkan kursor.

```
  if (ch==27) finish=1;
```

Keluar jika ESC ditekan.

```
    if (ch=='q') sy++;
    if (ch=='a') sy--;
    if (ch=='w') sx++;
    if (ch=='s') sx--;
```

Atur ukuran sprite sesuai dengan penekanan tombol keyboard q,a,w,s.

```
  }
  CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

Dengan dipanggilnya fungsi Install_Palette maka susunan palette yang ditampilkan di layar menjadi sesuai dengan gambar. Susunan palette ini dalam file disimpan dalam rupa array of R,G,B di mana masing-masing R,G,B berupa Byte. R,G,B ini menyimpan intensitas merah, hijau dan biru dari setiap cell tabel warna. Sistem palette ini sama persis seperti sistem palette pada unit kita pada seri terdahulu yaitu VGACAD.TPU.

Di bawah ini akan ditampilkan fungsi Install_Palette dan Refresh_Palette yang keduanya berada pada unit VGAPAL.CPP.

```
void Install_Palette(char * str)
{
        devPal=(char *)malloc(768);
        actPal=(char *)malloc(768);
```

Alokasikan memory sebesar 768 byte (R,G,B x 256 cell = 3 x 256 = 768) untuk devPal dan actPal. DevPal menyimpan palette standar yang akan kita gunakan, sedangkan actPal akan menyimpan palette yang sedang aktif.

```
        FILE * fpal;
        fpal=fopen(str,"rb");
        fread(devPal,768,1,fpal);
```

Baca palette dari file ke devPal.

```
        memcpy(actPal,devPal,768);
```

Kopikan palette dari devPal ke actPal sebagai palette yang akan kita

aktifkan ke layar monitor.

```
}

void Refresh_Palette()
{
        int a;
        for (a=0;a<256;a++)
        {
          Set_Palette(a,*(actPal+a*3),*(actPal+a*3+1),
          *(actPal+a*3+2));
        }
```

Aktifkan palette dari tabel actPal ke layar monitor dengan memanggil fungsi Set_Palette.

```
}

void Set_Palette(long no,char r, char g, char b)
{
        outp(0x3c8,no);
        outp(0x3c9,r);
        outp(0x3c9,g);
        outp(0x3c9,b);
```

Aktifkan palette dengan nomor cell no dan intensitas r,g,b. Caranya ialah dengan menggunakan port alamat $3c8 untuk mengirimkan nomor cell, sedangkan port $3c9 untuk mengirimkan intensitas r,g,b secara berturutan.

```
}
```

5.6 Teknik Efek Pencahayaan dan Pembayangan Gambar

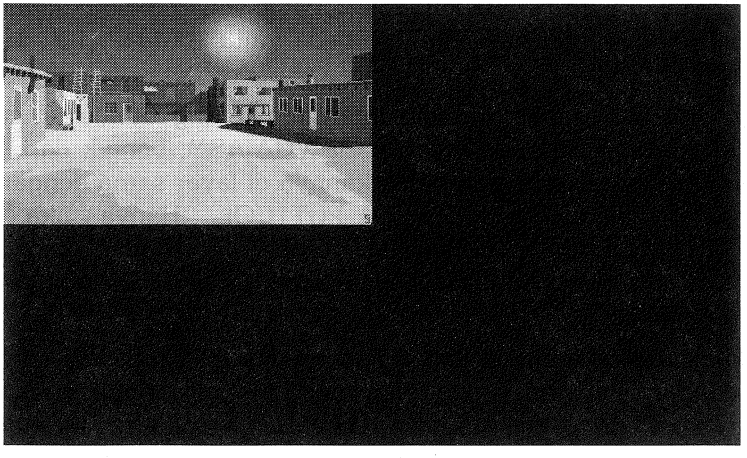
Fungsi VESA_Lightning

Fungsi VESA_Lightning ini digunakan untuk mencahayai suatu bagian dari gambar. Fungsi ini merupakan fungsi baru yang belum kita miliki pada VGACAD.TPU di seri sebelumnya.

Coba jalankan program LIGHT2.EXE, akan ditampilkan gambar background. Tekan ENTER sekali, akan ditampilkan gambar lingkaran. Lingkaran inilah yang nantinya kita gunakan sebagai bentuk dan intensitas terang dari lampu. Coba tekan ENTER sekali lagi. Di

sini kita dapat menggerakkan bola lampu dengan tombol kursor, juga kita dapat menekan tombol spasi untuk mengatur tingkat intensitas terang dari lampu. Lihat Gambar 5.19.

Sebenarnya teknik pencahayaan ini dilakukan dengan cara mengganti titik-titik yang akan kita terangi dengan warna yang lebih terang. Bagaimana cara mencari warna yang lebih terang dari suatu titik? Mudah, kita cukup mengalikan intensitas masing-masing R,G,B dengan suatu angka yang lebih besar dari 1. Misalnya 1.1 sehingga diperoleh intensitas terang 0.1 kali lipat. Misalnya, warna titik (misal warna nomor 10) yang akan diterangi memiliki intensitas warna R = 10, G = 15 dan B = 20. Maka setelah dikalikan dengan 1.1, intensitas warna yang baru ialah R = 11, G = 16 dan B = 22. Titik ini harus diganti dengan warna tersebut. Masalahnya ialah kita tidak dapat mengganti intensitas R,G,B dari tabel warna nomor 10 dengan intensitas yang baru, karena ada kemungkinan masih banyak titik yang lain yang menggunakan nomor warna ini.



Gambar 5.19 Teknik Pencahayaan dengan VGA.ASM

Jika kita ganti, maka titik-titik yang lain tersebut akan ditampilkan dengan warna yang baru juga. Untuk itu kita harus melakukan pencarian seluruh warna dalam tabel yang paling mendekati intensitas warna yang baru tersebut. Setelah kita temukan, kita akan memakai nomor warna tersebut sebagai warna titik setelah diterangi. Untuk selanjutnya untuk mempermudah pembahasan, warna baru ini akan kita namakan sebagai nama warna cahaya.

Khan dengan dilakukannya perkalian bilangan pecahan pada setiap titiknya juga ditambah dengan metode pencarian warna yang paling

menyerupai menjadikan animasi berjalan lambat? Benar... maka pemecahannya ialah kita akan menggunakan sebuah tabel yang memiliki 256 cell dan pada masing-masing cellnya berisikan nomor warna cahaya. Jadi pada cell nomor 10 isinya ialah nomor warna cahaya dari warna nomor 10. Sehingga pada saat kita akan menerangi titik yang berwarna 50, kita cukup mengambil warna cahaya dari titik nomor warna 50 ini dalam tabel dengan index 50.

Tabel ini dapat dibuat secara terpisah dalam program lain dan disimpan dalam sebuah file, sehingga setiap kali kita memerlukan tabel ini, tidak perlu dilakukan proses perhitungan dan pencarian lagi. Ingat, setiap susunan warna palette, memiliki tabel yang berbeda.

Masa kita puas dengan hanya menerapkan cahaya dengan kekuatan intensitas 1.1? Tentu tidak.... Kita ingin menciptakan efek pencahayaan di mana intensitas cahaya dapat kita atur baik 1.1, 1.2, 1.3 dan lain sebagainya.

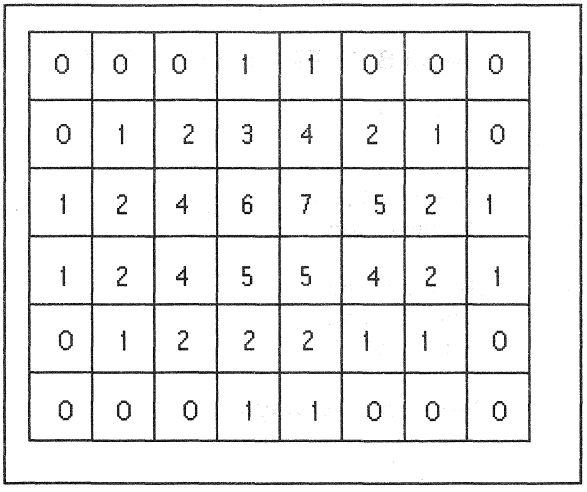
Untuk itu kita cukup menciptakan beberapa kolom dalam tabel cahaya (gunakan array dua dimensi). Pada setiap kolom digunakan variabel pengali intensitas yang berbeda yaitu 1.1, 1.2, 1.3, 1.4 dan seterusnya.

Dengan ini kita dapat memilih intensitas terang dari efek cahaya yang kita tampilkan pada layar. Pada VGA.ASM, bentuk dari lampu disimpan sebagai sprite dan peletakannya di layar sama dengan cara peletakan dengan fungsi VESA_PUT1COL. Hanya jika pada VESA_PUT1COL kita mengirimkan nomor warna yang akan ditampilkan, di sini kita mengirimkan intensitas terang dari cahaya yang akan diciptakan.

Jika kita melakukan penerangan pada setiap titik di bawah sprite cahaya dengan intensitas yang sama, maka pinggir-pinggir (perbatasan antara daerah yang kena cahaya dan tidak) sprite akan terlihat jelas dan menimbulkan efek yang janggal. Untuk itu kita harus mengurangi intensitas cahaya pada bagian pinggir sprite. Bagaimana caranya? Nggak sulit... kita cukup memanfaatkan warna titik-titik dalam sprite. Nomor warna ini kita manfaatkan sebagai intensitas cahaya dari titik tersebut. Sehingga setiap titik dalam sprite dapat memiliki intensitas cahaya yang berbeda. Dengan ini kita dapat menyediakan sprite cahaya dengan angka yang semakin kecil mendekati tepian dari sprite ini. Misalnya seperti pada Gambar 5.20.

Karena titik-titik pada sprite cahaya ini telah memiliki intensitas cahaya sendiri, maka intensitas cahaya yang dikirimkan lewat parameter akan digunakan sebagai penambah intensitas cahaya setiap

titik dalam sprite cahaya. Sehingga kita dapat mengganti tingkat terang dari sprite cahaya ini dengan memainkan parameter tadi. Hal inilah yang dilakukan pada program LIGHT2.EXE di atas.



Gambar 5.20 Sprite Cahaya yang Memiliki Intensitas Lebih Gelap pada Bagian Tepi

Sekarang, marilah kita tinjau listing program dari VESA_Lightning.

Pada VGA.ASM, tabel cahaya ini memiliki 32 kolom sehingga diperoleh 32 macam intensitas terang. Sehingga nomor warna yang digunakan untuk intensitas terang tiap titik dalam sprite juga berkisar antara 0 hingga 31.

```
VESA_Lightning          PROC        NEAR

        LOCAL    XWStart:DWord,XRStart:DWord,YWStart:DWord,
                 YRStart:DWord,XRead:DWORD,Widths:DWord,YRead
                   :DWord,Lengths:DWord

        push     ebp
        mov      ebp,esp
        sub      esp, 64
        ARG      PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                 BUFFER:DWORD,
                 TABLE:DWORD ,INTEN:DWORD
```

Kirimkan parameter PICBUFFER yang berisi alamat memory virtual screen yang akan diterangi. X1,Y1 ialah koordinat tempat sprite cahaya akan diterapkan. BUFFER ialah alamat memory sprite cahaya.

TABLE ialah alamat memory tempat tabel cahaya disimpan dan INTEN yang berisi intensitas terang dari sprite cahaya.

```
mov     esi, [BUFFER]
mov     edi, [PICBUFFER]
```

Isikan esi dan edi sebagai BUFFER dan PICBUFFER.

```
Xor     eax, eax
lodsw
mov     ecx, eax
Mov     [Lengths],eax
Mov     [XRead],eax
lodsw
Mov     [Widths],eax
Mov     [YRead],eax
Mov     [YRStart],0
```

Ambil panjang dan lebar sprite cahaya.

```
Mov     eax,[X1]
cmp     eax,S_L
jg      @LPI_Vin
Add     eax,[Lengths]
Sub     eax,1
Cmp     eax,0
jl      @LPI_Vin
Mov     eax,[Y1]
cmp     eax,S_W
jg      @LPI_Vin
Add     eax,[Widths]
Sub     eax,1
Cmp     eax,0
jl      @LPI_Vin
```

Batalkan pencahayaan jika sprite cahaya berada di luar layar.

```
Mov     eax,[Y1]
Mov     [YWStart],eax
Mov     [XRStart],0
Mov     eax,[X1]
Mov     [XWStart],eax
cmp     eax,0
Jge @LVESX1_Not_Negatives
Neg     eax
Mov     [XRStart],eax
Mov     [XWStart],0
Sub     [XRead],eax
```

Potong sprite terhadap sisi kiri layar.

```
@LVESX1_Not_Negatives:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @LVESX1_No_Right_Crops
  Sub     eax,[S_L]
  Sub     [XRead],eax
```

Potong sprite terhadap sisi kanan layar.

```
@LVESX1_No_Right_Crops:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @LVESNo_Top_Crops
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
```

Potong sprite terhadap sisi atas layar.

```
@LVESNo_Top_Crops:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @LVESNo_Bot_Crops
  Sub     eax,[S_W]
  Sub     [YRead],eax
```

Potong sprite terhadap sisi bawah layar.

```
@LVESNo_Bot_Crops:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax

  Mov     eax,[YRStart]
  Mul     [Lengths]
  Add     esi,eax
  Add     esi,[XRStart]
```

Hitung offset sprite titik pertama (kiri atas) dari sprite cahaya setelah dipotong.

```
Mov     ecx, [YRead]
```

Sediakan loop vertikal.

```
xor     eax,eax
xor     ebx,ebx
mov     edx,[INTEN]

@LVESvLoop1:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
```

Sediakan loop horizontal.

```
@LVESvLoop2:
   xor ebx,ebx
   Lodsb
   or al,al
   Jz @LVESvNo_Write
   Mov bl,[edi]
```

Deteksi, apakah warna titik = 0. Jika tidak, lakukan pencahayaan.

```
shl ebx,5
sub eax,edx
```

Caranya ialah dengan mengambil warna titik asli sebagai index dari tabel cahaya. Karena setiap cell dalam tabel cahaya sekarang berisi 32 byte (32 intensitas terang), maka kalikan index dengan 32 yaitu dengan menggeser kiri sejauh 5 bit.

Kemudian kurangi intensitas cahaya dari titik yang sedang diakses dengan intensitas cahaya yang dikirimkan lewat parameter. Sehingga sebenarnya intensitas yang dikirimkan lewat parameter ini ialah pengurang dari intensitas cahaya, bukanya penambah. Jadi intensitas 0 adalah maksimum, sedangkan 31 adalah intensitas minimum.

```
jge @noch
xor eax,eax
```

Jika intensitas hasil pengurangan lebih kecil dari 0, batasi harganya di 0.

```
@noch:
  add ebx,eax
  add ebx,[TABLE]
  mov al,[ebx]
```

Tambahkan intensitas cahaya akhir ke ebx yang berupa index ke tabel cahaya. Kemudian ambil nomor warna cahaya dari dalam tabel.

```
        Stosb
        dec edi
```

Letakkan warna cahaya ini ke virtual screen.

```
      @LVESvNo_Write:
        inc edi
      Loop @LVESvLoop2
      Pop esi
      Pop edi
      Add esi,[Lengths]
      Add edi,[A_SL]
      Pop ecx
      Loop @LVESvLoop1

@LPI_Vin:
      mov esp,ebp
      pop ebp
      ret
VESA_Lightning                ENDP
```

Mengenai pembuatan tabel cahaya ini, marilah kita melihat library VGAPAL.CPP pada fungsi Create_LightTable.

Fungsi ini digunakan untuk menciptakan tabel cahaya dari sebuah tabel palette.

```
void CreateLightTable(char * fname)
{
  int a,b,c,min,index,pm;
  float rr,rg,rb;
  char col;
  FILE * fpal;
  int  r1,g1,b1,r2,g2,b2,pr,pg,pb;
  float dr,dg,db;
  fpal=fopen(fname,"wb");
```

Buka file untuk tabel cahaya.

```
  for (a=0;a<256;a++)
  {
```

Lakukan loop untuk memproses seluruh nomor warna.

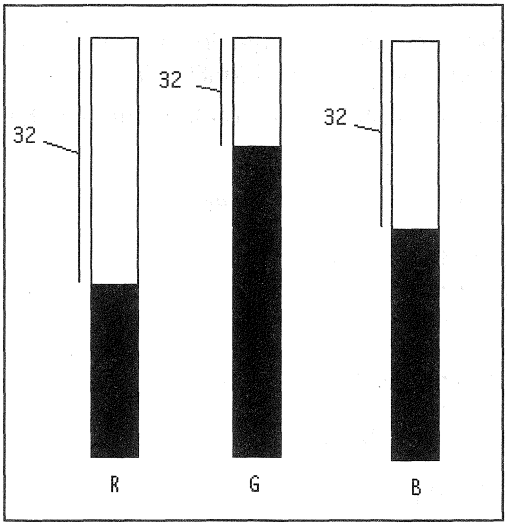
```
r1=*(actPal+a*3);
g1=*(actPal+a*3+1);
b1=*(actPal+a*3+2);
rr=(63-r1);
rg=(63-g1);
rb=(63-b1);
rr/=32;
rg/=32;
rb/=32;
```

Cari sebuah angka rr di mana angka ini merupakan hasil pembagian dengan 32 antara sisa intensitas R menuju ke maksimum (63). Jika R berharga 31 maka hargar rr = (63-31)/32 = 1. Hal ini ditujukan agar kita dapat membagi penambahan intensitas cahaya ke dalam 32 level intensitas terang untuk mengisi tabel cahaya.

Lakukan hal yang sama untuk intensitas G dan B.

```
dr=r1;dg=g1;db=b1;
```

Dr,Dg dan Db adalah intensitas R,G,B yang kita inginkan. Langkah selanjutnya ialah mencari warna yang memiliki R,G,B paling mirip dengan Dr, Dg dan Db.



Gambar 5.21 Pembagian Resolusi dari Masing-masing R,G,B Tiap Intensitas Warna

```
for (b=0;b<32;b++)
{
```

Lakukan loop untuk setiap intensitas cahaya.

```
index=0;
pr=*(actPal+0*3);
pg=*(actPal+0*3+1);
pb=*(actPal+0*3+2);
min=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
```

Anggap warna nomor 0 yang paling mirip dengan Dr, Dg dan Db. Hitung skor kemiripan yang dihitung dengan cara menghitung jumlah perbedaan intensitas R, G dan B dari warna 0 dan warna yang diinginkan.

```
for (c=0;c<256;c++)
{
  pr=*(actPal+c*3);
  pg=*(actPal+c*3+1);
  pb=*(actPal+c*3+2);
  pm=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
  if (pm<min)
  {
    min=pm;
    index=c;
  }
}
```

Lakukan loop untuk membandingkan harga pm dari setiap warna dengan yang saat ini sudah ditemukan. Jika ditemukan yang lebih mirip, maka tukar warna yang saat ini sudah ditemukan dengan warna yang baru.

```
col=index;
fwrite(&col,1,1,fpal);
dr+=rr;dg+=rg;db+=rb;
```

Tuliskan nomor warna dalam file dan tambah dr, dg, db dengan rr, rg, rb untuk mencari warna cahaya dengan intensitas berikutnya.

```
    }
  }
  fclose(fpal);
```

Tutup file tabel cahaya.

```
}
```

Nah, sekarang mari kita lihat penggunaan fungsi-fungsi di atas. Pertama-tama buka file LIGHT1.CPP. Program ini digunakan untuk me-

manggil fungsi CreateLightTable dalam membuat palette cahaya untuk program LIGHT2.CPP.

PENJELASAN PROGRAM

```
/*    ------------------------------------------------------
Nama File : LIGHT1.CPP
Definisi  : Program untuk membuat tabel cahaya.
Programmer: Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
```

Definisikan VGAPAL.CPP dan header filenya VGAPAL.H.

```
int main()
{
  OpenVGA256(MOD400);
  Install_Palette("dev.pal");
```

Buka mode graphic 640 x 400 256 warna dan install palette dalam file DEV.PAL.

```
  Refresh_Palette();
  CreateLightTable("light.tab");
  CloseVGA256();
```

Aktifkan palette kemudian panggil fungsi CreateLightTable dengan nama file yang akan dibuat yatu LIGHT.TAB.

Tutup mode graphic.

```
  return 1;
}

/*  ------------------------------------------------------Nama
File : LIGHT2.CPP
Definisi  : Program mencoba efek cahaya
Programmer: Agustinus Nalwan
------------------------------------------------------------ */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
```

Definisikan library VGAPAL.CPP dan header filenya VGAPAL.H.

```
char ch;
char * Buffer;
char * Picture;
char * Back;
```

Deklarasikan variabel untuk menampung virtual screen Buffer dan Back, variabel untuk menampung sprite cahaya yaitu Picture.

```
char * LightTable;
```

Deklarasikan variabel untuk menampung tabel cahaya LightTable.

```
int inten=0;
```

Intensitas cahaya awal diset = 0 (maksimum).

```
void LoadPicture(char * fname, char * Buff)
{
  FILE * fpic;
  int x,y;
  char col;
  fpic=fopen(fname,"rb");
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
      fread(&col,1,1,fpic);
      VESA_PutPixel(Buff,x,y,col);
    }
  }
  fclose(fpic);
```

Tampilkan gambar ke virtual screen.

```
}

int main()
{
```

```
int finish=0,x=160,y=100;
```

Letakkan sprite pada koordinat (160,100).

```
Buffer=(char *)malloc(MEM400);
Back=(char *)malloc(MEM400);
Picture=(char *)malloc(15000);
LightTable=(char *)malloc(8192);
```

Alokasikan memory untuk Buffer, Back dan Picture. Kemudian alokasikan juga memory untuk LightTable sebesar 8192 Byte (256 x 32).

```
OpenVGA256(MOD400);
Install_Palette("dev.pal");
Install_Lightning(LightTable,"light.tab");
```

Buka mode graphic, gunakan file DEV.PAL sebagai palette standard. Buka tabel LIGHT.TAB dan masukkan ke dalam memory LightTable.

```
Refresh_Palette();
VESA_ClearDouble(Back,0);
LoadPicture("town.pic",Back);
```

Aktifkan palette, hapus virtual screen Back dan isikan dengan gambar dari file TOWN.PIC.

```
VESA_MoveToScreen(Back);
ch=getch();
```

Tampilkan virtual screen ke layar dan tunggu penekanan tombol keyboard.

```
LoadPicture("sprite.pic",Buffer);
VESA_GetImage(Buffer,1,1,79,65,Picture);
VESA_MoveToScreen(Buffer);
ch=getch();
```

Baca file SPRITE.PIC yang berisi sprite cahaya dan masukkan ke dalam alamat memory Picture.

```
VESA_ClearDouble(Buffer,0);
while (!finish)
{
  VESA_MoveDouble(Back,Buffer);
  VESA_Lightning(Buffer,x,y,Picture,LightTable,inten);
```

Tampilkan sprite cahaya ke virtual screen buffer dengan fungsi VESA_Lightning pada koordinat X,Y dan dengan menggunakan tabel LightTabel dan dengan intensitas cahaya inten.

```
VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen buffer ke layar monitor.

```
while (!kbhit())
{
}
ch=getch();
if (ch==72) y--;
if (ch==80) y++;
if (ch==75) x--;
if (ch==77) x++;
```

Gerakkan sprite sesuai dengan penenkanan tombol keyboard.

```
if (ch==' ') inten++;
```

Jika tombol spasi ditekan tambah variabel inten (kurangi intensitas dari cahaya).

```
if (ch==27) finish=1;
```

Keluar dari program jika ESC ditekan.

```
if (inten>31) inten=0;
```

Jika intensitas cahaya melebihi 31 maka kembalikan ke 0.

```
  }
  CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

Fungsi VESA_Llightning

Fungsi ini merupakan mode pencerminan dari fungsi VESA_Lightning. Letak perbedaannya dengan fungsi VESA_Lightning ialah terdapat pada bagian perhitungan offset sprite yang terbalik seperti yang telah kita bahas pada fungsi VESA_LPutimage.

```
VESA_LLightning         PROC        NEAR

        LOCAL   XWStart:DWord,XRStart:DWord,YWStart:DWord,
                YRStart:DWord,XRead:DWORD,Widths:DWord,YRead
                  :DWord,Lengths:DWord

        push    ebp
        mov     ebp,esp
        sub     esp, 64
        ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                BUFFER:DWORD,TABLE:DWORD ,INTEN:DWORD

        mov     esi, [BUFFER]
        mov     edi, [PICBUFFER]

        Xor     eax, eax
        lodsw
        mov     ecx, eax
        Mov     [Lengths],eax
        Mov     [XRead],eax
        lodsw
        Mov     [Widths],eax
        Mov     [YRead],eax
        Mov     [YRStart],0
```

Baca panjang dan lebar sprite.

```
        Mov     eax,[X1]
        cmp     eax,S_L
        jg      @MrLPI_Vin
        Add     eax,[Lengths]
        Sub     eax,1
        Cmp     eax,0
        jl      @MrLPI_Vin
        Mov     eax,[Y1]
        cmp     eax,S_W
        jg      @MrLPI_Vin
        Add     eax,[Widths]
        Sub     eax,1
        Cmp     eax,0
        jl      @MrLPI_Vin
```

Batalkan pencahayaan jika sprite berada di luar layar.

```
        Mov     eax,[Y1]
        Mov     [YWStart],eax
        Mov     [XRStart],0
        Mov     eax,[X1]
```

```
  Mov     [XWStart],eax
  cmp     eax,0
Jge @MrLVESX1_Not_Negatives
  Neg     eax
  Mov     [XRStart],eax
  Mov     [XWStart],0
  Sub     [XRead],eax
```

Potong sprite terhadap batas kiri layar.

```
@MrLVESX1_Not_Negatives:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @MrLVESX1_No_Right_Crops
  Sub     eax,[S_L]
  Sub     [XRead],eax
```

Potong sprite terhadap batas kanan layar.

```
@MrLVESX1_No_Right_Crops:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @MrLVESNo_Top_Crops
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
```

Potong sprite terhadap batas atas layar.

```
@MrLVESNo_Top_Crops:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @MrLVESNo_Bot_Crops
  Sub     eax,[S_W]
  Sub     [YRead],eax
```

Potong sprite terhadap batas bawah layar.

```
@MrLVESNo_Bot_Crops:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax
```

Hitung offset memory dari titik pertama ditampilkannya sprite pada virtual screen.

```
Mov     eax,[YRStart]
Mul     [Lengths]
Add     esi,eax
Add     esi,[Lengths]
Sub     esi,[XRStart]
dec     esi
```

Hitung offset sprite.

```
Mov     ecx, [YRead]
xor     eax,eax
xor     ebx,ebx
mov     edx,[INTEN]

@MrLVESvLoop1:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
@MrLVESvLoop2:
   xor ebx,ebx
   mov al,[esi]
   dec esi
```

Kurangi esi untuk memproses titik horizontal sebelumnya.

```
   or al,al
   Jz @MrLVESvNo_Write
```

Deteksi, apakah titik berwarna 0. Jika tidak, lanjutkan proses pencahayaan.

```
   Mov bl,[edi]
   shl ebx,5
   sub eax,edx
   jge @Mrnoch
   xor eax,eax
 @Mrnoch:
   add ebx,eax
   add ebx,[TABLE]
   mov al,[ebx]
```

Ambil warna cahaya dari tabel cahaya.

```
          Stosb
          dec edi
        @MrLVESvNo_Write:
          inc edi
      Loop @MrLVESvLoop2
      Pop esi
      Pop edi
      Add esi,[Lengths]
      Add edi,[A_SL]
      Pop ecx
      Loop @MrLVESvLoop1

@MrLPI_Vin:
      mov esp,ebp
      pop ebp
      ret

VESA_LLightning                ENDP
```

Fungsi selanjutnya yang mirip sekali dengan fungsi pencahayaan ini ialah fungsi pembuatan bayangan. Kalau pada pencahayaan, dilakukan penerangan terhadap bagian gambar, pada pembayangan, dilakukan penggelapan terhadap bagian gambar.

Yang menarik di sini ialah, fungsi pembayangan ini dilakukan dengan menggunakan fungsi yang sama dengan fungsi pencahayaan yaitu VESA_Lightning. Kenapa? Karena sistem pembayangan cara kerjanya sama dengan fungsi pencahayaan, hanya saja kalau pada pencahayaan dilakukan penambahan intensitas warna dari titik-titik yang terkena cahaya, pada pembayangan dilakukan pengurangan intensitas warna dari titik-titik yang kena bayangan. Sehingga yang perlu kita ganti ialah tabelnya.

Mari kita lihat bagaimana implementasi dari tabel fungsi pembayangan ini.

```
void CreateShadowTable(char * fname)
{
  int a,b,c,min,index,pm;
  float rr,rg,rb;
  char col;
  FILE * fpal;
  int  r1,g1,b1,r2,g2,b2,pr,pg,pb;
  float dr,dg,db;
  fpal=fopen(fname,"wb");
```

Buka file untuk tabel bayangan.

```
for (a=0;a<256;a++)
{
  r1=*(actPal+a*3);
  g1=*(actPal+a*3+1);
  b1=*(actPal+a*3+2);
  rr=r1;rg=g1;rb=b1;
  rr/=32;
  rg/=32;
  rb/=32;
  dr=r1;dg=g1;db=b1;
```

Kalau dulu dr, dg dan db dihitung dengan mengurangi 63 ke intensitas warna (karena warna tujuan ialah 63 yaitu putih terang), sekarang cukup langsung membagi intensitas warna dengan 32 karena warna tujuan ialah warna 0 yaitu hitam.

```
  for (b=0;b<32;b++)
  {
    index=0;
```

Lakukan loop untuk memproses seluruh intensitas.

```
    pr=*(actPal+0*3);
    pg=*(actPal+0*3+1);
    pb=*(actPal+0*3+2);
    min=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
```

Anggap warna nomor 0 yang paling mirip dengan warna yang kita inginkan.

```
    for (c=0;c<256;c++)
    {
      pr=*(actPal+c*3);
      pg=*(actPal+c*3+1);
      pb=*(actPal+c*3+2);
      pm=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
      if (pm<min)
      {
        min=pm;
        index=c;
      }
    }
    col=index;
```

Cari warna yang paling mirip dari 256 warna.

```
    fwrite(&col,1,1,fpal);
    dr-=rr;dg-=rg;db-=rb;
```

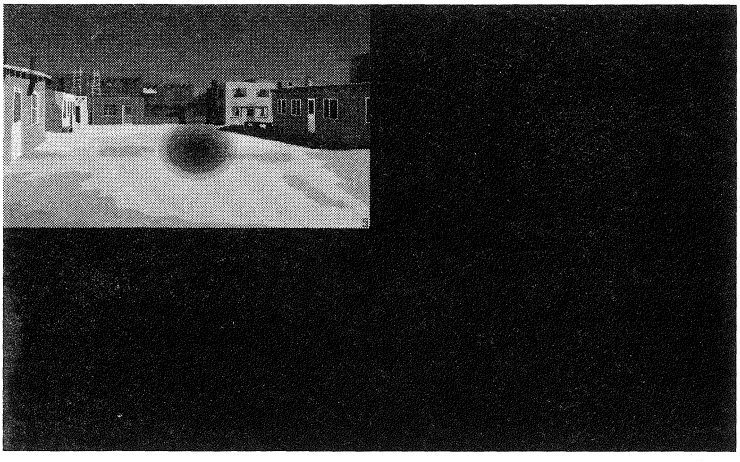
Tuliskan nomor warna ke dalam file dan kurangi dr, dg dan db untuk memproses intensitas berikutnya (semakin mendekati 0).

```
    }
  }
  fclose(fpal);
```

Tutup file bayangan.

```
}
```

Progam SHADOW1.EXE digunakan untuk menciptakan tabel bayangan. Untuk mencoba fungsi pembayangan ini jalankan program SHADOW2.EXE.



Gambar 5.22 Output dari Fungsi Pembayangan

Di sinipun kita dapat menggerakkan sprite bayangan dengan menekan tombol kursor. Juga kita dapat mengatur intensitas bayangan dengan menekan tombol spasi.

PENJELASAN PROGRAM

```
/* ------------------------------------------------------
Nama File : SHADOW2.CPP
Definisi  : Program mencoba efek bayangan
Programmer: Agustinus Nalwan
------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
```

```
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"

char ch;
char * Buffer;
char * Picture;
char * Back;
char * ShadowTable;

int inten=0;

void LoadPicture(char * fname, char * Buff)
{
  FILE * fpic;
  int x,y;
  char col;
  fpic=fopen(fname,"rb");
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
      fread(&col,1,1,fpic);
      VESA_PutPixel(Buff,x,y,col);
    }
  }
  fclose(fpic);
}
```

Tampilkan gambar pada virtual screen.

```
int main()
{
  int finish=0,x=160,y=100;
  Buffer=(char *)malloc(MEM400);
  Back=(char *)malloc(MEM400);
  Picture=(char *)malloc(15000);
  ShadowTable=(char *)malloc(8192);
```

Alokasikan memory untuk virtual screen, tabel bayangan dan sprite.

```
  OpenVGA256(MOD400);
  Install_Palette("dev.pal");
  Install_Shadow(ShadowTable,"shadow.tab");
  Refresh_Palette();
```

Buka mode graphic 640 x 400 256 warna. Install dan aktifkan palette DEV.PAL. Install tabel bayangan SHADOW.TAB.

```
VESA_ClearDouble(Back,0);
LoadPicture("town.pic",Back);
VESA_MoveToScreen(Back);
ch=getch();
```

Hapus virtual screen Back, tampilkan gambar dari file TOWN.PIC ke virtual screen Back dan tampilkan ke layar.

```
LoadPicture("sprite.pic",Buffer);
VESA_GetImage(Buffer,1,1,79,65,Picture);
VESA_MoveToScreen(Buffer);
ch=getch();
```

Tampilkan gambar SPRITE.PIC ke virtual screen Buffer, rekam sprite ke Picture dan tampilkan virtual screen ke layar.

```
VESA_ClearDouble(Buffer,0);
while (!finish)
{
  VESA_MoveDouble(Back,Buffer);
  VESA_Lightning(Buffer,x,y,Picture,ShadowTable,inten);
```

Tampilkan sprite bayangan pada virtual screen dengan koordinat (x,y) dan intensitas inten.

```
  VESA_MoveToScreen(Buffer);
  while (!kbhit())
  {
  }
  ch=getch();
  if (ch==72) y--;
  if (ch==80) y++;
  if (ch==75) x--;
  if (ch==77) x++;
  if (ch==' ') inten++;
  if (ch==27) finish=1;
  if (inten>31) inten=0;
```

Gerakkan sprite sesuai gerakan kursor dan atur intensitas bayangan sesuai dengan penekanan tombol spasi. Keluar dari program jika ESC ditekan.

```
}
CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

Fungsi berikutnya yang masih mirip dengan fungsi pencahayaan dan pembayangan ialah efek perubahan warna (color mask). Efek ini jika diterapkan akan menimbulkan kesan pencahayaan berwarna. Jika biasanya dengan VESA_Lightning dilakukan pencahayaan warna putih, maka di sini kita dapat menentukan warna cahaya sesuka kita (misalnya gambar terkena cahaya merah/kuning). Sama halnya seperti fungsi pembayangan, di sini kita cukup mengganti tabel. Fungsi yang digunakan tetap VESA_Lightning.

Yuk, kita lihat bagaimana pembuatan tabel cahayanya.

```
void CreateMaskTable(char * fname,int desr,int desg,int
desb)
{
  int a,b,c,min,index,pm;
  float rr,rg,rb;
  char col;
  FILE * fpal;
  int  r1,g1,b1,r2,g2,b2,pr,pg,pb;
  float dr,dg,db;
  fpal=fopen(fname,"wb");
```

Buka file tabel cahaya.

```
  for (a=0;a<256;a++)
  {
    r1=*(actPal+a*3);
    g1=*(actPal+a*3+1);
    b1=*(actPal+a*3+2);
    rr=desr-r1;rg=desg-g1;rb=desb-b1;
    rr/=32;
    rg/=32;
    rb/=32;
    dr=r1;dg=g1;db=b1;
```

Karena sekarang warna tujuan dapat diinputkan lewat parameter yaitu desr, desg dan desb maka kita cukup mengurangi intensitas r,g,b dengan parameter tersebut kemudian membaginya dengan 32.

```
    for (b=0;b<32;b++)
    {
```

Lakukan loop untuk memproses seluruh intensitas warna.

```
      index=0;
      pr=*(actPal+0*3);
      pg=*(actPal+0*3+1);
      pb=*(actPal+0*3+2);
      min=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
      for (c=0;c<256;c++)
      {
        pr=*(actPal+c*3);
        pg=*(actPal+c*3+1);
        pb=*(actPal+c*3+2);
        pm=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
        if (pm<min)
        {
          min=pm;
          index=c;
        }
      }
      col=index;
```

Cari warna yang paling mirip dengan warna yang diinginkan.

```
      fwrite(&col,1,1,fpal);
      dr+=rr;dg+=rg;db+=rb;
```

Tulis ke file dan lompat ke intensitas selanjutnya.

```
    }
  }
  fclose(fpal);
}
```

Pembuatan tabel cahaya berwarna ini dilakukan oleh file MASK1.EXE. Di sini akan diciptakan tabel cahaya merah (R = 63, G = 0 dan B = 0).

Untuk membuktikan fungsi ini, coba jalankan program MASK2.EXE.

WOW!!! Cahaya yang jatuh sekarang berwarna merah.

PENJELASAN PROGRAM

Karena program di bawah ini mirip sekali dengan kedua program yang telah kita bahas di atas, maka kita lompati saja pembahasan dari listing program ini.

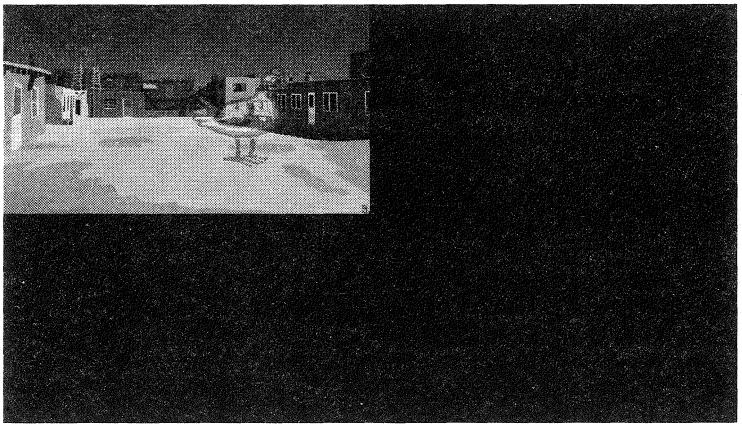
```
/* ------------------------------------------------------------
Nama File : MASK2.CPP
Definisi  : Program mencoba efek cahaya merah
Programmer: Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"

char ch;
char * Buffer;
char * Picture;
char * Back;
char * MaskTable;
int inten=0;

void LoadPicture(char * fname, char * Buff)
{
  FILE * fpic;
  int x,y;
  char col;
  fpic=fopen(fname,"rb");
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
      fread(&col,1,1,fpic);
      VESA_PutPixel(Buff,x,y,col);
    }
  }
  fclose(fpic);
}
int main()
{
  int finish=0,x=160,y=100;
  Buffer=(char *)malloc(MEM400);
  Back=(char *)malloc(MEM400);
  Picture=(char *)malloc(15000);
  MaskTable=(char *)malloc(8192);

  OpenVGA256(MOD400);
```

```
  Install_Palette("dev.pal");
  Install_Lightning(MaskTable,"redmask.tab");
  Refresh_Palette();
  VESA_ClearDouble(Back,0);
  LoadPicture("town.pic",Back);
  VESA_MoveToScreen(Back);
  ch=getch();

  LoadPicture("sprite.pic",Buffer);
  VESA_GetImage(Buffer,1,1,79,65,Picture);
  VESA_MoveToScreen(Buffer);
  ch=getch();

  VESA_ClearDouble(Buffer,0);
  while (!finish)
  {
    VESA_MoveDouble(Back,Buffer);
    VESA_Lightning(Buffer,x,y,Picture,MaskTable,inten);
    VESA_MoveToScreen(Buffer);
    while (!kbhit())
    {
    }
    ch=getch();
    if (ch==72) y--;
    if (ch==80) y++;
    if (ch==75) x--;
    if (ch==77) x++;
    if (ch==' ') inten++;
    if (ch==27) finish=1;
    if (inten>31) inten=0;
  }
  CloseVGA256();
  return 1;
}
```

Nah, yang perlu kita perhatikan dalam fungsi-fungsi pencahayaan di atas ialah kita harus memperhatikan susunan warna palette yang kita gunakan. Karena dalam perancangan tabel cahaya merah akan dilakukan pencarian seluruh warna (0-256) jika terkena cahaya merah dengan 32 macam kekuatan intensitas, maka warna-warna hasil pencarian tersebut haruslah tersedia dalam susunan palette kita (atau paling tidak ada warna-warna yang mirip), karena jika tidak, hasil pencarian akan menggunakan warna yang benar-benar tidak mirip karena tidak ditemukan satu pun yang mirip sehingga menimbulkan efek yang buruk.

Fungsi VESA_PutTransparant

Fungsi ini digunakan untuk menampilkan sprite dengan mode transparan. Coba jalankan program TRANS2.EXE. Keren kan? Gerak-gerakkan sprite dengan menekan kursor. Bagian bawah sprite masih terlihat samar-samar karena mode yang digunakan ialah transparan.



Gambar 5.23 Output dari Fungsi Transparansi

Sebenarnya teknik transparansi ini dilakukan juga dengan menggunakan tabel. Jika pada pencahayaan digunakan tabel untuk mencari warna cahaya dari setiap warna, di sini digunakan tabel dua dimensi di mana kedua indeks tabel merupakan nomor warna 0-255. Dengan ini, maka untuk mencari warna hasil transparansi antara warna 40 (warna titik pada sprite yang akan diletakkan) dan warna 80 (warna titik pada virtual screen di bawah titik yang akan diletakkan), caranya ialah dengan mengambil angka di dalam tabel dengan indeks (40,80). Warna hasil transparansi ini dihitung dengan melakukan rata-rata dari intensitas antara dua warna yang akan dicari transparansinya.

Sehingga ukuran tabel ialah 256 x 256 = 65536 Bytes.

```
VESA_PutTransparant  PROC        NEAR

        LOCAL   XWStart:DWord,XRStart:DWord,YWStart:DWord,
                YRStart:DWord,XRead:DWORD,Widths:DWord,YRead
                :DWord,Lengths:DWord

        push    ebp
        mov     ebp,esp
        sub     esp, 64
```

```
ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
        BUFFER:DWORD,TABLE:DWORD
```

Parameter yang dikirimkan ialah PICBUFFER yang berisi alamat virtual screen yang akan digambari. (X1,Y1) ialah koordinat sprite yang akan diletakkan dan BUFFER yaitu sprite yang akan diletakkan. TABLE ialah tabel transparansi yang akan digunakan.

```
mov     esi, [BUFFER]
mov     edi, [PICBUFFER]

Xor     eax, eax
lodsw
mov     ecx, eax
Mov     [Lengths],eax
Mov     [XRead],eax
lodsw
Mov     [Widths],eax
Mov     [YRead],eax
Mov     [YRStart],0
```

Ambil panjang dan lebar sprite.

```
Mov     eax,[X1]
cmp     eax,S_L
jg      @PLPI_Vin
Add     eax,[Lengths]
Sub     eax,1
Cmp     eax,0
jl      @PLPI_Vin
Mov     eax,[Y1]
cmp     eax,S_W
jg      @PLPI_Vin
Add     eax,[Widths]
Sub     eax,1
Cmp     eax,0
jl      @PLPI_Vin
```

Batalkan penggambaran sprite jika sprite berada di luar layar.

```
Mov     eax,[Y1]
Mov     [YWStart],eax
Mov     [XRStart],0
Mov     eax,[X1]
Mov     [XWStart],eax
cmp     eax,0
Jge @PLVESX1_Not_Negatives
Neg     eax
```

```
    Mov     [XRStart],eax
    Mov     [XWStart],0
    Sub     [XRead],eax
@PLVESX1_Not_Negatives:
    Mov     eax,[X1]
    Add     eax,[Lengths]
    Dec     eax
    Cmp     eax,[S_L]
    Jle     @PLVESX1_No_Right_Crops
    Sub     eax,[S_L]
    Sub     [XRead],eax
@PLVESX1_No_Right_Crops:
    Mov     eax,[Y1]
    Cmp     eax,0
    Jge     @PLVESNo_Top_Crops
    Neg     eax
    Mov     [YWStart],0
    Mov     [YRStart],eax
    Sub     [YRead],eax
@PLVESNo_Top_Crops:
    Mov     eax,[Y1]
    Add     eax,[Widths]
    Dec     eax
    Cmp     eax,[S_W]
    Jle     @PLVESNo_Bot_Crops
    Sub     eax,[S_W]
    Sub     [YRead],eax
```

Lakukan pemotongan terhadap batas kiri, kanan, atas dan bawah layar.

```
@PLVESNo_Bot_Crops:
    Add     edi,[XWStart]
    Mov     eax,[YWStart]
    mul     [A_SL]
    Add     edi,eax
```

Hitung offset memory dari titik pertama sprite pada virtual screen.

```
    Mov     eax,[YRStart]
    Mul     [Lengths]
    Add     esi,eax
    Add     esi,[XRStart]
```

Hitung offset sprite dari titik pertama sprite yang akan diletakkan.

```
    Mov     ecx, [YRead]
    xor     eax,eax
```

```
  xor     ebx,ebx
  xor     edx,edx
@PLVESvLoop1:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
@PLVESvLoop2:
   xor ebx,ebx
   Lodsb
   or al,al
   Jz @PLVESvNo_Write
```

Deteksi, apakah warna yang akan diletakkan ialah warna 0. Jika tidak, teruskan proses peletakan titik.

```
   Mov bl,[edi]
   shl ebx,8
```

Ambil warna titik yang akan diletakkan, gunakan sebagai indeks pertama (indeks vertikal) dari tabel. Karena tabelnya ialah tabel dua dimensi di mana masing-masing indeks vertikal dibagi menjadi 256 indeks horizontal, maka kita harus mengalikan indeks vertikal dengan 256 yaitu dengan menggeser ke kiri 8 bit.

```
@Pnoch:
   add ebx,eax
   add ebx,[TABLE]
```

Kemudian tambahkan hasil perhitungan indeks dengan indeks horizontal yang diperoleh dari warna titik di bawah titik yang akan diletakkan.

```
   mov al,[ebx]
   Stosb
```

Ambil warna hasil transparansi dalam tabel dan letakkan ke virtual screen.

```
   dec edi
 @PLVESvNo_Write:
   inc edi
Loop @PLVESvLoop2
Pop esi
Pop edi
Add esi,[Lengths]
Add edi,[A_SL]
```

```
        Pop ecx
        Loop @PLVESvLoop1

@PLPI_Vin:
        mov esp,ebp
        pop ebp
        ret

VESA_PutTransparant             ENDP
```

Sederhana bukan? Prosesnya sama persis seperti apa yang telah kita gunakan dalam merakit VGACAD.TPU dengan fungsi transparan.

Sekarang perhatikan fungsi untuk pembuatan tabel transparansi dalam file VGAPAL.CPP. Pembuatannya dilakukan dengan menghitung hasil intensitas rata-rata antara setiap pasang warna kemudian warna yang paling mirip dengan hasil perhitungan diisikan ke dalam tabel dengan indeks yaitu nomor dari kedua pasang warna tadi.

```
void CreateTransTable(char * fname)
{
  int a,b,c,min,index,pm;
  char col;
  FILE * fpal;
  int  r1,g1,b1,r2,g2,b2,dr,dg,db,pr,pg,pb;
  fpal=fopen(fname,"wb");
```

Buka file untuk tabel transparansi.

```
  for (a=0;a<256;a++)
  {
    r1=*(actPal+a*3);
    g1=*(actPal+a*3+1);
    b1=*(actPal+a*3+2);
```

Ambil intensitas r, g dan b dari warna pertama.

```
    for (b=0;b<256;b++)
    {
```

Lakukan loop untuk mencari pasangan antara warna pertama dengan seluruh warna yang ada.

```
      r2=*(actPal+b*3);
      g2=*(actPal+b*3+1);
      b2=*(actPal+b*3+2);
```

Ambil intensitas warna kedua.

```
dr=(r1+r2)/2;
dg=(g1+g2)/2;
db=(b1+b2)/2;
```

Lakukan rata-rata antara kedua warna.

```
index=0;
pr=*(actPal+0*3);
pg=*(actPal+0*3+1);
pb=*(actPal+0*3+2);
min=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
for (c=0;c<256;c++)
{
  pr=*(actPal+c*3);
  pg=*(actPal+c*3+1);
  pb=*(actPal+c*3+2);
  pm=abs(pr-dr)+abs(pg-dg)+abs(pb-db);
  if (pm<min)
  {
    min=pm;
    index=c;
  }
}
col=index;
```

Cari warna yang paling mirip dengan hasil rata-rata.

```
fwrite(&col,1,1,fpal);
```

Tuliskan ke tabel transparansi.

```
    }
  }
  fclose(fpal);
}
```

Pembuatan tabel transparansi ini dilakukan oleh program TRANS1.EXE.

Nah, berikutnya kita akan menggabungkan fungsi transparansi dengan fungsi peletakan sprite satu warna. Tentu saja hal ini sangat mudah dilakukan karena prosesnya sama hanya pada saat peletakan titik, warna sprite yang digunakan sebagai indeks diganti dengan warna yang telah dikirimkan lewat parameter.

```
VESA_PutTransparant1Col  PROC          NEAR

        LOCAL   XWStart:DWord,XRStart:DWord,YWStart:DWord,
```

```
YRStart:DWord,XRead:DWORD,Widths:DWord,YRead
:DWord,Lengths:DWord

push    ebp
mov     ebp,esp
sub     esp, 64
ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
        BUFFER:DWORD,TABLE:DWORD, COLOR:DWORD
```

Kirimkan juga warna yang akan digunakan sebagai pengganti warna titik sprite.

```
mov     esi, [BUFFER]
mov     edi, [PICBUFFER]

Xor     eax, eax
lodsw
mov     ecx, eax
Mov     [Lengths],eax
Mov     [XRead],eax
lodsw
Mov     [Widths],eax
Mov     [YRead],eax
Mov     [YRStart],0
```

Ambil panjang dan lebar sprite.

```
Mov     eax,[X1]
cmp     eax,S_L
jg      @P1PLPI_Vin
Add     eax,[Lengths]
Sub     eax,1
Cmp     eax,0
jl      @P1PLPI_Vin
Mov     eax,[Y1]
cmp     eax,S_W
jg      @P1PLPI_Vin
Add     eax,[Widths]
Sub     eax,1
Cmp     eax,0
jl      @P1PLPI_Vin
```

Batalkan penggambaran sprite jika sprite berada di luar layar.

```
Mov     eax,[Y1]
Mov     [YWStart],eax
Mov     [XRStart],0
Mov     eax,[X1]
```

```
  Mov     [XWStart],eax
  cmp     eax,0
Jge @P1PLVESX1_Not_Negatives
  Neg     eax
  Mov     [XRStart],eax
  Mov     [XWStart],0
  Sub     [XRead],eax
@P1PLVESX1_Not_Negatives:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @P1PLVESX1_No_Right_Crops
  Sub     eax,[S_L]
  Sub     [XRead],eax
@P1PLVESX1_No_Right_Crops:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @P1PLVESNo_Top_Crops
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
@P1PLVESNo_Top_Crops:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @P1PLVESNo_Bot_Crops
  Sub     eax,[S_W]
  Sub     [YRead],eax
```

Lakukan pemotongan sprite terhadap batas-batas layar.

```
@P1PLVESNo_Bot_Crops:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax
```

Hitung offset memory dari titik kiri atas sprite pada virtual screen.

```
  Mov     eax,[YRStart]
  Mul     [Lengths]
  Add     esi,eax
  Add     esi,[XRStart]
```

Hitung offset sprite dari titik kiri atas sprite.

```
        Mov     ecx, [YRead]
        xor     eax,eax
        xor     ebx,ebx
        xor     edx,edx
      @P1PLVESvLoop1:
        Push ecx
        Push edi
        Push esi
        Mov ecx, [XRead]
      @P1PLVESvLoop2:
         xor ebx,ebx
         Lodsb
         or al,al
         Jz @P1PLVESvNo_Write
         Mov eax,[COLOR]
```

Di sini eax diganti dengan warna yang dikirimkan lewat parameter. Sehingga diperoleh efek peletakan titik satu warna saja.

```
         Mov bl,[edi]
         shl ebx,8
      @P1Pnoch:
         add ebx,eax
         add ebx,[TABLE]
         mov al,[ebx]

         Stosb
         dec edi
       @P1PLVESvNo_Write:
         inc edi
      Loop @P1PLVESvLoop2
      Pop esi
      Pop edi
      Add esi,[Lengths]
      Add edi,[A_SL]
      Pop ecx
      Loop @P1PLVESvLoop1

@P1PLPI_Vin:
      mov esp,ebp
      pop ebp
      ret

VESA_PutTransparant1Col          ENDP
```

Fungsi VESA_LPutTransparant

Fungsi ini merupakan mode pencerminan dari fungsi VESA_PutTransparant. Sama seperti mode-mode pencerminan yang lainnya, bagian yang diganti ialah bagian perhitungan offset sprite dan arah pembacaan titik sprite secara horizontal.

```
VESA_LPutTransparant  PROC        NEAR

      LOCAL     XWStart:DWord,XRStart:DWord,YWStart:DWord,
                YRStart:DWord,XRead:DWORD,Widths:DWord,
                YRead:DWord,Lengths:DWord

        push    ebp
        mov     ebp,esp
        sub     esp, 64
        ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                BUFFER:DWORD,TABLE:DWORD

        mov     esi, [BUFFER]
        mov     edi, [PICBUFFER]

        Xor     eax, eax
        lodsw
        mov     ecx, eax
        Mov     [Lengths],eax
        Mov     [XRead],eax
        lodsw
        Mov     [Widths],eax
        Mov     [YRead],eax
        Mov     [YRStart],0

        Mov     eax,[X1]
        cmp     eax,S_L
        jg      @MrPLPI_Vin
        Add     eax,[Lengths]
        Sub     eax,1
        Cmp     eax,0
        jl      @MrPLPI_Vin
        Mov     eax,[Y1]
        cmp     eax,S_W
        jg      @MrPLPI_Vin
        Add     eax,[Widths]
        Sub     eax,1
        Cmp     eax,0
        jl      @MrPLPI_Vin
```

```
  Mov     eax,[Y1]
  Mov     [YWStart],eax
  Mov     [XRStart],0
  Mov     eax,[X1]
  Mov     [XWStart],eax
  cmp     eax,0
Jge @MrPLVESX1_Not_Negatives
  Neg     eax
  Mov     [XRStart],eax
  Mov     [XWStart],0
  Sub     [XRead],eax
@MrPLVESX1_Not_Negatives:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @MrPLVESX1_No_Right_Crops
  Sub     eax,[S_L]
  Sub     [XRead],eax
@MrPLVESX1_No_Right_Crops:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @MrPLVESNo_Top_Crops
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
@MrPLVESNo_Top_Crops:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @MrPLVESNo_Bot_Crops
  Sub     eax,[S_W]
  Sub     [YRead],eax
@MrPLVESNo_Bot_Crops:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax

  Mov     eax,[YRStart]
  Mul     [Lengths]
  Add     esi,eax
  Add     esi,[Lengths]
  dec     esi
  Sub     esi,[XRStart]
```

Proses perhitungan offset sprite dilakukan terbalik karena sprite akan digambarkan terbalik.

```
        Mov     ecx, [YRead]
        xor     eax,eax
        xor     ebx,ebx
        xor     edx,edx
      @MrPLVESvLoop1:
        Push ecx
        Push edi
        Push esi
        Mov ecx, [XRead]
      @MrPLVESvLoop2:
         xor ebx,ebx
         mov al,[esi]
         dec esi
         or al,al
         Jz @MrPLVESvNo_Write
         Mov bl,[edi]
         shl ebx,8
      @MrPnoch:
         add ebx,eax
         add ebx,[TABLE]
         mov al,[ebx]

         Stosb
         dec edi
       @MrPLVESvNo_Write:
         inc edi
      Loop @MrPLVESvLoop2
      Pop esi
      Pop edi
      Add esi,[Lengths]
      Add edi,[A_SL]
      Pop ecx
      Loop @MrPLVESvLoop1

@MrPLPI_Vin:
      mov esp,ebp
      pop ebp
      ret

VESA_LPutTransparant          ENDP
```

Fungsi VESA_LPutTransparant1Col

Fungsi ini merupakan mode pencerminan dari fungsi

VESA_PutTransparant1Col.

```
VESA_LPutTransparant1Col  PROC        NEAR

        LOCAL     XWStart:DWord,XRStart:DWord,YWStart:DWord,
                  YRStart:DWord,XRead:DWORD,Widths:DWord,YRead
                  :DWord,Lengths:DWord

          push     ebp
          mov      ebp,esp
          sub      esp, 64
          ARG      PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                   BUFFER:DWORD,TABLE:DWORD, COLOR:DWORD

          mov      esi, [BUFFER]
          mov      edi, [PICBUFFER]

          Xor      eax, eax
          lodsw
          mov      ecx, eax
          Mov      [Lengths],eax
          Mov      [XRead],eax
          lodsw
          Mov      [Widths],eax
          Mov      [YRead],eax
          Mov      [YRStart],0

          Mov      eax,[X1]
          cmp      eax,S_L
          jg       @MrP1PLPI_Vin
          Add      eax,[Lengths]
          Sub      eax,1
          Cmp      eax,0
          jl       @MrP1PLPI_Vin
          Mov      eax,[Y1]
          cmp      eax,S_W
          jg       @MrP1PLPI_Vin
          Add      eax,[Widths]
          Sub      eax,1
          Cmp      eax,0
          jl       @MrP1PLPI_Vin

          Mov      eax,[Y1]
          Mov      [YWStart],eax
          Mov      [XRStart],0
          Mov      eax,[X1]
          Mov      [XWStart],eax
```

```
  cmp     eax,0
Jge @MrP1PLVESX1_Not_Negatives
  Neg     eax
  Mov     [XRStart],eax
  Mov     [XWStart],0
  Sub     [XRead],eax
@MrP1PLVESX1_Not_Negatives:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[S_L]
  Jle     @MrP1PLVESX1_No_Right_Crops
  Sub     eax,[S_L]
  Sub     [XRead],eax
@MrP1PLVESX1_No_Right_Crops:
  Mov     eax,[Y1]
  Cmp     eax,0
  Jge     @MrP1PLVESNo_Top_Crops
  Neg     eax
  Mov     [YWStart],0
  Mov     [YRStart],eax
  Sub     [YRead],eax
@MrP1PLVESNo_Top_Crops:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[S_W]
  Jle     @MrP1PLVESNo_Bot_Crops
  Sub     eax,[S_W]
  Sub     [YRead],eax
@MrP1PLVESNo_Bot_Crops:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]
  mul     [A_SL]
  Add     edi,eax

  Mov     eax,[YRStart]
  Mul     [Lengths]
  Add     esi,eax
  Add     esi,[Lengths]
  dec     esi
  Sub     esi,[XRStart]
```

Perhitungan offset sprite dilakukan terbalik.

```
  Mov     ecx, [YRead]
  xor     eax,eax
```

```
        xor     ebx,ebx
        xor     edx,edx
      @MrP1PLVESvLoop1:
        Push ecx
        Push edi
        Push esi
        Mov ecx, [XRead]
      @MrP1PLVESvLoop2:
         xor ebx,ebx
         mov al,[esi]
         dec esi
         or al,al
         Jz @MrP1PLVESvNo_Write
         Mov eax,[COLOR]
         Mov bl,[edi]
         shl ebx,8
      @MrP1Pnoch:
         add ebx,eax
         add ebx,[TABLE]
         mov al,[ebx]

         Stosb
         dec edi
       @MrP1PLVESvNo_Write:
         inc edi
      Loop @MrP1PLVESvLoop2
      Pop esi
      Pop edi
      Add esi,[Lengths]
      Add edi,[A_SL]
      Pop ecx
      Loop @MrP1PLVESvLoop1

@MrP1PLPI_Vin:
      mov esp,ebp
      pop ebp
      ret

VESA_LPutTransparant1Col          ENDP
```

Fungsi-fungsi di bawah ini merupakan fungsi-fungsi VGA.ASM yang menggunakan mode viewport. Adapun tidak seluruh fungsi diciptakan mode viewportnya.

```
VESA_VWP_PutPixel         PROC NEAR

        push    ebp
```

```
        mov     ebp,esp
        ARG     BUFFER:DWORD, X:DWORD, Y:DWORD, COL:BYTE
        mov     eax,[Y]
        cmp     eax,[V_Y2]
        jg      @VPPFinish
        cmp     eax,[V_Y1]
        jl      @VPPFinish
        mov     eax,[X]
        cmp     eax,[V_X2]
        jg      @VPPFinish
        cmp     eax,[V_X1]
        jl      @VPPFinish

        mov     eax,[Y]
        mul     [A_SL]
        add     eax,[X]
        mov     edi,[BUFFER]
        mov     cl,COL
        mov     BYTE PTR [edi+eax],cl
      @VPPFinish:
        mov     esp,ebp
        pop     ebp
        ret

VESA_VWP_PutPixel       ENDP


VESA_VWP_Put3D   PROC NEAR

        LOCAL   XWStart:DWord,YWStart:DWord,Length:DWord,
                Width:DWord,XRead:DWord,YRead:DWord,XFixed:
                DWord,Yfixed:DWord,XFStart:DWord,YFStart:
                DWord,RX:DWord,RY:DWord=AUTO_SIZE

        Push    ebp
        mov     ebp,esp
        sub     esp, AUTO_SIZE
        ARG     PICBUFF:DWORD,X:DWORD,Y:DWORD,XL:DWORD,
                YL:DWORD,BUFFER:DWORD

        mov     esi,[BUFFER]
        mov     edi,[PICBUFF]

        xor     eax,eax
        Lodsw
        Mov     [Length],eax
```

```
    shl     eax,16
    Mov     ebx,eax
    Xor     eDx,eDx

    Div     [XL]

    Mov     [Rx],eax
    Mov     eax,[XL]
    Mov     [XRead],eax

    xor     eax,eax
    Lodsw
    Mov     [Width],eAx
    shl     eax,16
    Mov     ebx,eax
    Xor     eDx,eDx
    Div     [YL]
    Mov     [Ry],eAx
    Mov     eAx,eBx
    Xor     eDx,eDx
    Mov     eax,[YL]
    Mov     [YRead],eAx
    Mov     [XFixed],0
    Mov     [YFixed],0
    Mov     [XFStart],0
    Mov     [YFStart],0
    Mov     eAx,[Y]
    Mov     [YWStart],eAx

    Mov     eAx,[X]
    Mov     [XWStart],eAx

    sub eax,[V_X1]
    Cmp eAx,0
Jge @VWPP3_X1_Not_Negative
    Neg eAx
    Mov ebx,[V_X1]
    Mov XWStart,ebx
    Sub XRead,eAx
    Mul Rx
    Mov XFStart,eAx
@VWPP3_X1_Not_Negative:
    Mov eAx,XWStart
    Add eAx,XRead
    Dec eAx
    Cmp eAx,[V_X2]
Jle @VWPP3_X1_No_Right_Crop
```

```
    Sub eAx,[V_X2]
    Sub XRead,eAx
@VWPP3_X1_No_Right_Crop:
    Mov eAx,Y
    Sub eax,[V_Y1]
    Cmp eAx,0
Jge @VWPP3_No_Top_Crop
    Neg eAx
    Mov ebx,[V_Y1]
    Mov YWStart,ebx
    Sub YRead,eAx
    Mul Ry
    Mov YFStart,eAx
@VWPP3_No_Top_Crop:
    Mov eAx,YWStart
    Add eAx,YRead
    Cmp eAx,[V_Y2]
Jle @VWPP3_No_Bot_Crop
    Sub eAx,[V_Y2]
    Sub YRead,eAx
@VWPP3_No_Bot_Crop:
    Add edi,[XWStart]
    Mov eAx,[YWStart]
    mul [A_SL]
    Add edi,eAx

    Mov eCx,[YRead]
    Mov eAx,[YFStart]
    Mov [YFixed],eAx
@VWPP3_Loop1:
  Push eCx
  Push edi
  Push esi
  Mov eAx,[YFixed]
  shr eax,16
  Mul [Length]
  Add esi,eAx

  Mov eCx,[XRead]
  Mov eAx,[XFStart]
  Mov [XFixed],eAx
  xor ebx,ebx
  Mov eBx,[Rx]
  @VWPP3_Loop2:
     Mov eAx,[XFixed]
     shr eax,16
     Push esi
```

```
          Add esi,eAx

          Lodsb
          Pop esi
          Add [XFixed],eBx
          or al,al
          Jz @VWPP3_No_Write
          Stosb
          dec edi
        @VWPP3_No_Write:
          inc edi
       Loop @VWPP3_Loop2
       Pop esi
       Pop edi
       xor ebx,ebx
       Mov eBx,[Ry]
       Add [YFixed],eBx
       Add edi,[A_SL]
       Pop eCx
       Loop @VWPP3_Loop1

       mov  esp,ebp
       pop  ebp

       ret
VESA_VWP_Put3D   ENDP

VESA_VWP_Line        PROC NEAR
       LOCAL     IFlag:DWORD,CalX:DWORD,CalY:DWORD,
                 DelX:DWORD,DelY:DWORD,Dsx:DWORD,Dsy:DWORD,
                 Rx:DWORD,Ry:DWORD, Fixed:DWORD,YFixed:DWORD,
                 TLoop:DWORD=AUTO_SIZE

       push      ebp
       mov       ebp,esp
       sub       esp,AUTO_SIZE
       ARG       PICBUFF:DWORD,X1:DWORD,Y1:DWORD,X2:DWORD,
                 Y2:DWORD,COL:BYTE

       mov       esi,[PICBUFF]

       mov       ebx,[Y2]
       sub       ebx,[Y1]
       jge       @VWPLno_swap
       mov       ecx,[Y2]
       mov       ebx,[Y1]
       mov       [Y2],ebx
```

```
  mov       [Y1],ecx
  sub       ebx,ecx
  mov       [Dsy],ebx
  mov       ecx,[X2]
  mov       ebx,[X1]
  mov       [X2],ebx
  mov       [X1],ecx
  jmp       @VWPLNo_Add
@VWPLno_swap:
  mov       [Dsy],ebx
@VWPLNo_Add:
  mov       eax,[X1]
  sub       eax,[X2]
  push      eax
  cmp       eax,0
  jge       @VWPL_Case2
  neg       eax
@VWPL_Case2:
  mov       [Dsx],eax
  mov       eax,[Dsx]
  sub       eax,[Dsy]
  jge       @VWPXFixed

  mov       [Ry],10000h
  mov       eax,[Dsx]
  shl       eax,16
  xor       edx,edx
  cmp       [Dsy],0
  jz        @terminate
  div       [Dsy]
  mov       [Rx],eax
  mov       eax,[Dsy]
  jmp       @VWPTest_Crop
@VWPXFixed:
  mov       [Rx],10000h
  mov       eax,[Dsy]
  shl       eax,16
  xor       edx,edx
  cmp       [Dsx],0
  jz        @terminate
  div       [Dsx]
  mov       [Ry],eax
  mov       eax,[Dsx]
@VWPTest_Crop:
  mov      [IFlag],0
  mov      [TLoop],eax
  mov      [XFixed],0
```

```
    mov       [YFixed],0

    mov       eax,[V_X2]
    cmp       [X1],eax
    jle       @VVNext1
    cmp       [X2],eax
    jle       @VVNext1
    jmp       @VWPFinish

@VVNext1:
    mov       eax,[V_X1]
    cmp       [X1],eax
    jge       @VVNext2
    cmp       [X2],eax
    jge       @VVNext2
    jmp       @VWPFinish

@VVNext2:
    mov       eax,[V_Y2]
    cmp       [Y1],eax
    jg        @VWPFinish
    mov       eax,[V_Y1]
    cmp       [Y2],eax
    jl        @VWPFinish

    mov       eax,[Y1]
    sub       eax,[V_Y1]
    neg       eax
    jle       @VWPTest_Side
    mov       [delY],eax
    mul       [Rx]
    xor       edx,edx
    cmp       [Ry],0
    jz        @terminate
    div       [Ry]
    mov       [CalX],eax
    pop       ecx
    push      ecx
    cmp       ecx,0
    jle       @VWPNo_neg
    neg       eax
@VWPNo_neg:
    add       eax,[X1]
    cmp       eax,[V_X2]
    jg        @VWPTest_Side
    cmp       eax,[V_X1]
    jl        @VWPTest_Side
```

```
  mov       [IFlag],1
  cmp       [Rx],10000h
  jne       @VWPUse_Ry

  mov       eax,[CalX]
  mov       ebx,eax
  sub       [TLoop],eax
  shl       eax,16
  mov       [XFixed],eax
  mov       eax,ebx
  mul       [Ry]
  mov       [YFixed],eax
  jmp       @VWPTest_Bottom
@VWPUse_Ry:

  mov       eax,[delY]
  mov       ebx,eax
  sub       [TLoop],eax
  shl       eax,16
  mov       [YFixed],eax
  mov       eax,ebx
  mul       [Rx]
  mov       [XFixed],eax
  jmp       @VWPTest_Bottom
@VWPTest_Side:

  pop       ecx
  push      ecx
  cmp       ecx,0
  jl        @VWPTo_Right

  mov       eax,[X1]
  sub       eax,[V_X2]
  jle       @VWPTest_Bottom
  mov       [delX],eax
  jmp       @VWPCont_Side


@VWPTo_Right:
  mov       eax,[X1]
  sub       eax,[V_X1]
  neg       eax
  jle       @VWPTest_Bottom
  mov       [delX],eax

@VWPCont_Side:
```

```
  mul       [Ry]
  xor       edx,edx
  cmp       [Rx],0
  jz        @terminate
  div       [Rx]
  mov       [CalY],eax
  add       eax,[Y1]
  cmp       eax,[V_Y1]
  jl        @VWPTest_Bottom
  cmp       eax,[V_Y2]
  jg        @VWPTest_Bottom

  mov       [IFlag],1
  cmp       [Rx],10000h
  jne       @VWPUse_RyS

  mov       eax,[delX]
  sub       [TLoop],eax
  shl       eax,16
  mov       [XFixed],eax
  mov       eax,[delX]
  mul       [Ry]
  mov       [YFixed],eax
  jmp       @VWPTest_Bottom
@VWPUse_RyS:

  mov       eax,[CalY]
  sub       [TLoop],eax
  shl       eax,16
  mov       [YFixed],eax
  mov       eax,[CalY]
  mul       [Rx]
  mov       [XFixed],eax
@VWPTest_Bottom:

  cmp       [IFlag],0
  jne       @VWPTest_Bottom_Crop
  mov       eax,[V_X2]
  cmp       [X1],eax
  jg        @VWPFinish
  mov       eax,[V_X1]
  cmp       [X1],eax
  jl        @VWPFinish
  mov       eax,[V_Y1]
  cmp       [Y1],eax
  jl        @VWPFinish
  mov       eax,[V_Y2]
```

```
  cmp       [Y1],eax
  jg        @VWPFinish
@VWPTest_Bottom_Crop:

  mov       eax,[Y2]
  sub       eax,[V_Y2]
  jle       @VWPTest_Final

  mov       [delY],eax
  mul       [Rx]
  xor       edx,edx
  cmp       [Ry],0
  jz        @terminate
  div       [Ry]
  mov       [CalX],eax

  pop       ecx
  push      ecx
  cmp       ecx,0
  jge       @VWPBot_Add

  neg       eax
@VWPBot_Add:

  add       eax,[X2]
  cmp       eax,[V_X2]
  jg        @VWPTest_Final
  cmp       eax,[V_X1]
  jl        @VWPTest_Final

  cmp       [Rx],10000h
  jne       @VWPUse_RyB

  mov       eax,[CalX]
  jmp       @VWPCont_Bottom
@VWPUse_RyB:

  mov       eax,[delY]

@VWPCont_Bottom:
  sub       [TLoop],eax
  jmp       @VWPTest_Dir

@VWPTest_Final:

  pop       ecx
  push      ecx
```

```
  cmp       ecx,0
  jge       @VWPTo_RightF

  mov       eax,[X2]
  sub       eax,[V_X2]
  jle       @VWPTest_Dir
  mov       [delX],eax
  jmp       @VWPCont_Final

@VWPTo_RightF:

  mov       eax,[X2]
  sub       eax,[V_X1]
  neg       eax
  jle       @VWPTest_Dir
  mov       [delX],eax

@VWPCont_Final:

  mul       [Ry]
  xor       edx,edx
  cmp       [Rx],0
  jz        @terminate
  div       [Rx]
  mov       [CalY],eax
  mov       ebx,[Y2]
  sub       ebx,eax
  cmp       ebx,[V_Y1]
  jl        @VWPTest_Dir
  cmp       ebx,[V_Y2]
  jg        @VWPTest_Dir

  mov       ebx,[delX]
  cmp       [Rx],10000h
  je        @VWPUse_RxF
  mov       ebx,[CalY]
@VWPUse_RxF:
  sub       [TLoop],ebx

@VWPTest_Dir:

  pop       eax
  push      eax
  cmp       eax,0
  jl        @VWPDraw_Line

  neg       [Rx]
```

```
      neg       [XFixed]

    @VWPDraw_Line:
      mov       eax,[Y1]
      mul       [A_SL]
      add       eax,[X1]
      add       esi,eax

      mov       ecx,[TLoop]
      inc       ecx
      cmp       ecx,0
      jle        @VWPFinish
    @VWPLLoop:
      mov       eax,[YFixed]
      shr       eax,16
      adc       eax,0
      mul       [A_SL]

      mov       ebx,[XFixed]
      sar       ebx,16
      adc       ebx,0
      add       eax,ebx
      mov       dl,[Col]
      mov       BYTE PTR [esi+eax],dl
      mov       eax,[Rx]
      add       [XFixed],eax
      mov       eax,[Ry]
      add       [YFixed],eax
    loop @VWPLLoop

    @VWPFinish:
    @terminate:
      pop       eax
      mov       esp,ebp
      pop       ebp
      ret

VESA_VWP_Line       ENDP


VESA_VWP_PutImage         PROC       NEAR

        LOCAL   XWStart:DWord,XRStart:DWord,YWStart:DWord,
                YRStart:DWord,XRead:DWORD,Widths:DWord,YRead
                :DWord,Lengths:DWord

        push    ebp
```

```
  mov     ebp,esp
  sub     esp, 64
  ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
          BUFFER:DWORD

  mov     esi, [BUFFER]
  mov     edi, [PICBUFFER]

  Xor     eax, eax
  lodsw
  mov     ecx, eax
  Mov     [Lengths],eax
  Mov     [XRead],eax
  lodsw
  Mov     [Widths],eax
  Mov     [YRead],eax
  Mov     [YRStart],0
  Mov     eax,[Y1]
  Mov     [YWStart],eax
  Mov     [XRStart],0
  Mov     eax,[X1]
  Mov     [XWStart],eax
  sub     eax,[V_X1]
  cmp     eax,0
Jge @VWPX1_Not_Negatives
  Neg     eax
  Mov     [XRStart],eax
  Mov     ebx,[V_X1]
  Mov     [XWStart],ebx
  Sub     [XRead],eax
@VWPX1_Not_Negatives:
  Mov     eax,[X1]
  Add     eax,[Lengths]
  Dec     eax
  Cmp     eax,[V_X2]
  Jle     @VWPX1_No_Right_Crops
  Sub     eax,[V_X2]
  Sub     [XRead],eax
@VWPX1_No_Right_Crops:
  Mov     eax,[Y1]
  sub     eax,[V_Y1]
  Cmp     eax,0
  Jge     @VWPNo_Top_Crops
  Neg     eax
  mov     ebx,[V_Y1]
  Mov     [YWStart],ebx
  Mov     [YRStart],eax
```

```
  Sub     [YRead],eax
@VWPNo_Top_Crops:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[V_Y2]
  Jle     @VWPNo_Bot_Crops
  Sub     eax,[V_Y2]
  Sub     [YRead],eax
@VWPNo_Bot_Crops:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]

  mul     [A_SL]
  Add     edi,eax

  Mov     eax,[YRStart]
  Mul     [Lengths]
  Add     esi,eax
  Add     esi,[XRStart]

  Mov     ecx, [YRead]
@VWPvLoop1:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
@VWPvLoop2:
   Lodsb
   or al,al
   Jz @VWPvNo_Write
   Stosb
   dec edi
 @VWPvNo_Write:
   inc edi
Loop @VWPvLoop2
Pop esi
Pop edi
Add esi,[Lengths]
Add edi,[A_SL]
Pop ecx
Loop @VWPvLoop1

mov esp,ebp
pop ebp

ret
```

```
VESA_VWP_PutImage                    ENDP

VESA_VWP_Put1Col          PROC        NEAR

        LOCAL   XWStart:DWord,XRStart:DWord,YWStart:DWord,
                YRStart:DWord,XRead:DWORD,Widths:DWord,YRead
                :DWord,Lengths:DWord=AUTO_SIZE

        push    ebp
        mov     ebp,esp
        sub     esp, AUTO_SIZE
        ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                BUFFER:DWORD,COL:BYTE

        mov     esi, [BUFFER]
        mov     edi, [PICBUFFER]

        Xor     eax, eax
        lodsw
        mov     ecx, eax
        Mov     [Lengths],eax
        Mov     [XRead],eax
        lodsw
        Mov     [Widths],eax
        Mov     [YRead],eax
        Mov     [YRStart],0
        Mov     eax,[Y1]
        Mov     [YWStart],eax
        Mov     [XRStart],0
        Mov     eax,[X1]
        Mov     [XWStart],eax
        sub     eax,[V_X1]
        cmp     eax,0
      Jge @VWPX1_Not_Negativescol
        Neg     eax
        Mov     [XRStart],eax
        Mov     ebx,[V_X1]
        Mov     [XWStart],ebx
        Sub     [XRead],eax
      @VWPX1_Not_Negativescol:
        Mov     eax,[X1]
        Add     eax,[Lengths]
        Dec     eax
        Cmp     eax,[V_X2]
        Jle     @VWPX1_No_Right_Cropscol
        Sub     eax,[V_X2]
        Sub     [XRead],eax
```

```
@VWPX1_No_Right_Cropscol:
  Mov     eax,[Y1]
  sub     eax,[V_Y1]
  Cmp     eax,0
  Jge     @VWPNo_Top_Cropscol
  Neg     eax
  mov     ebx,[V_Y1]
  Mov     [YWStart],ebx
  Mov     [YRStart],eax
  Sub     [YRead],eax
@VWPNo_Top_Cropscol:
  Mov     eax,[Y1]
  Add     eax,[Widths]
  Dec     eax
  Cmp     eax,[V_Y2]
  Jle     @VWPNo_Bot_Cropscol
  Sub     eax,[V_Y2]
  Sub     [YRead],eax
@VWPNo_Bot_Cropscol:
  Add     edi,[XWStart]
  Mov     eax,[YWStart]

  mul     [A_SL]
  Add     edi,eax

  Mov     eax,[YRStart]
  Mul     [Lengths]
  Add     esi,eax
  Add     esi,[XRStart]

  Mov     ecx, [YRead]
  Mov     dl,[Col]
@VWPvLoop1col:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
@VWPvLoop2col:
   Lodsb
   or al,al
   Jz @VWPvNo_Writecol
   mov al,dl
   Stosb
   dec edi
 @VWPvNo_Writecol:
   inc edi
Loop @VWPvLoop2col
```

```
        Pop esi
        Pop edi
        Add esi,[Lengths]
        Add edi,[A_SL]
        Pop ecx
        Loop @VWPvLoop1col

        mov esp,ebp
        pop ebp
        ret

VESA_VWP_Put1Col                    ENDP


ENDS

END
```

BAB 6

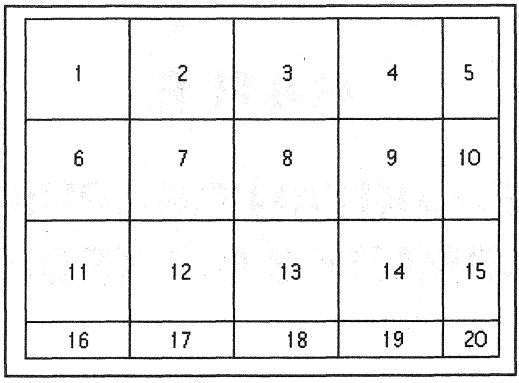
PERAKITAN GRAPHICS LIBRARY 8 BIT KEDUA

6.1 Fungsi Parallax Scrolling

Pada bab ini kita akan membahas fungsi-fungsi library yang berkenaan dengan parallax scrolling.

Parallax scrolling ialah fasilitas yang memungkinkan kita menciptakan sebuah background yang jauh lebih besar dari ukuran layar dan pada suatu saat kita dapat menampilkan bagian dari background tadi pada layar monitor. Parallax scrolling ini digunakan pada game-game petualangan seperti DIABLO, Ultima dan lain sebagainya. Sebenarnya fungsi ini dirakit dengan cara yang sama pada VGA-CAD.TPU. Hanya saja di sini digunakan 32 bit programming sehingga ukuran background yang dapat ditampilkan maksimum = 8000 x 5000 (hampir sama dengan 100 kali lipat ukuran layar pada mode graphic 800 x 600).

Sebenarnya bagaimana cara menciptakan background sebesar ini? Bagi para pembaca yang belum mengikuti seri Pemrograman Animasi dan Game Profesional 2 tentu belum mengetahui. Mari kita bahas secara singkat. Background sebesar ini sebenarnya dibagi-bagi menjadi sprite-sprite kecil berukuran 80 x 50 titik. Tiap sprite ini disimpan dalam array satu dimensi seperti pada Gambar 6.1.



Gambar 6.1 Penyimpanan Sprite Untuk Parallax Scroll

Gambar sebesar ini dapat Anda rancang pada software graphic apa saja asal mendukung format PCX. Simpan gambar di file dalam format PCX 8 bit kemudian panggil program PCXCON8.EXE yang merupakan program konversi dari PCX ke format parallax scroll. Hasil konversi akan berupa dua buah file dengan ekstensi ATR (berisi informasi dari gambar) dan dengan ekstensi SCR (berisi potongan-potongan sprite).

Kita tidak akan membahas format file PCX maupun pembuatan program PCXCON8.EXE ini karena keduanya telah kita bahas pada seri Pemrograman Animasi dan Game Profesional 2.

Adapun cara pemanggilan program PCXCON8.EXE ini ialah dengan menyebutkan nama PCX file (dengan ekstensi) yang akan dikonversikan. Hasil konversi akan berupa file BK1.SCR dan BK1.ATR.

Penggunaan memory untuk background yang besar ini juga sangat boros. Bayangkan, untuk gambar seukuran 1000 x 1000 membutuhkan memory 1 Mega Byte. Untuk itu kita akan melakukan kompresi pada setiap potongan sprite. Metode kompresi yang digunakan ialah RLC/RLL (Run Length Coding) seperti yang kita gunakan pada VGA-CAD.TPU. Jika Anda masih ingat, format dari RLL ini ialah digunakannya dua buah paket data.

Paket data pertama ialah diawali dengan angka bukan 0 yang menunjukkan jumlah titik yang bersebelahan yang memiliki warna yang sama, sedangkan warna titik-titik ini disimpan pada byte berikutnya. Paket jenis kedua diawali dengan angka 0 dan diikuti dengan angka yang menunjukkan jumlah titik dalam paket ini, kemudian diikuti dengan nomor-nomor warna titik-titik tersebut. Paket jenis ini digunakan untuk menyimpan titik-titik berdekatan yang memiliki warna bervariasi.

Misalkan kita memiliki data sebagai berikut:

1 1 1 2 4 8 2 4 5 5 5 5 5 5 5 5 5 2 3 5 6 6

Akan dikompres menjadi:

3 1 0 5 2 4 8 2 4 10 5 0 3 2 3 5 2 6

Yang berarti:

- 3 kali perulangan angka 1
- Data tidak dikompres sejumlah 5 yaitu 2 4 8 2 4
- 10 kali perulangan angka 5
- Data yang tidak dikompres sejumlah 3 yaitu 2 3 5.
- 2 kali perulangan angka 6.

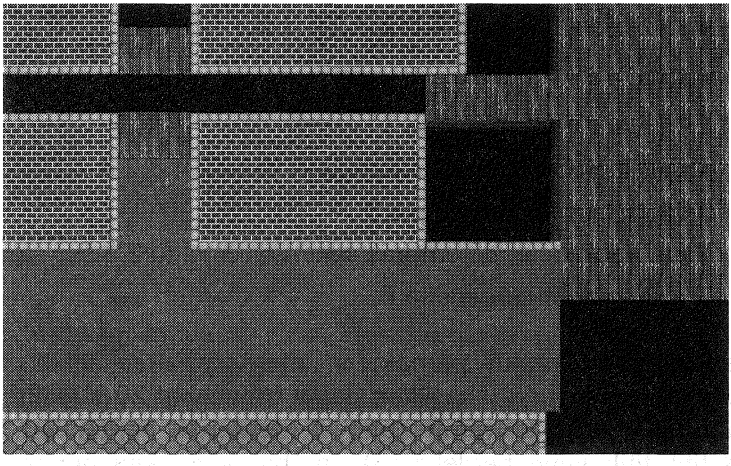
Dengan cara ini maka pemakaian memory akan lebih hemat. Adapun kompresi yang dihasilkan dapat mencapai 20 % untuk gambar normal. Semakin rumit (banyak terdapat pergantian warna antara titik-titik yang berdekatan) gambar yang kita gunakan, semakin jelek hasil dari kompresi.

Dalam implementasi nantinya, seluruh potongan sprite akan dikompresi secara terpisah. Hasil kompres inilah yang nantinya akan disimpan dalam array. Dan pada saat penampilan di layar, cukup dilakukan pengecekan potongan sprite mana yang akan ditampilkan pada layar, dan pada masing-masing sprite lakukan pembukaan kompres dari potongan sprite tersebut, kemudian tampilkan ke layar dengan menggunakan fungsi yang mirip VESA_PutImage. Sederhana bukan?

Satu hal lagi yang akan kita lakukan ialah mengoptimasi kecepatan penggambaran. Karena gambar sebesar ini tentu saja sangat lambat untuk ditampilkan pada layar monitor yang memiliki resolusi tinggi. Mengingat kadang beberapa bagian dari gambar background memiliki warna polos maka rugi jika kita harus membuka kompres dan melakukan penampilan potongan sprite tersebut. Kita cukup menggantikan operasi ini dengan fungsi VESA_Block yang berkali-kali lipat lebih cepat. Untuk itu kita akan mendeklarasikan variabel baru yaitu array yang berisikan angka yang akan berharga bukan 0 jika potongan sprite yang memiliki indeks yang sama dengan cell array tersebut akan diganti dengan fungsi VESA_Block. Sedangkan warna yang bukan 0 tadi ialah warna yang akan digunakan sebagai warna balok.

Coba kita jalankan program TSCR.EXE. Gerakkan tombol keyboard untuk menggeser background. Bagaimana? Pada mode resolusi

tinggi, tampilan terlihat jauh lebih bagus dibandingkan dengan mode resolusi 320 x 200 256 warna. Output program ini ditampilkan pada Gambar 6.2.



Gambar 6.2 Output dari Program Pencoba Parallax Scrolling

Mari kita tinjau listing program TSCR.CPP.

```
/* ------------------------------------------------------------
Nama File   : TSCR.CPP
Definisi    : Program mencoba parallax scroll
Programmer  : Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
#include "scroll.cpp"
#include "scroll.h"
```

Definisikan library SCROLL.CPP dengan header filenya SCROLL.H.

```
char ch;
char * Buffer;
char * Temp;
```

```
int main()
{
  int finish=0,x=4,y=4;
  Buffer=(char *)malloc(MEM400);
  Temp=(char *)malloc(MEM400);
```

Alokasikan memory untuk virtual screen Buffer. Alokasikan memory untuk buffer proses pembukaan kompres. Pada proses pembukaan kompres, kita harus menyediakan tempat kosong di mana hasil kompres akan diletakkan. Ukuran tempat kosong ini paling sedikit harus seukuran dengan gambar asli sebelum dikompres. Di sini disediakan memory seukuran dengan virtual screen.

```
  MaxX=639; MaxY=399;
```

Set kedua variabel MaxX dan MaxY dengan koordinat maksimum horizontal dan vertikal. Kedua variabel ini digunakan oleh fungsi-fungsi parallax scrolling sebagai patokan penampilan potongan sprite.

```
  OpenVGA256(MOD400);
  Install_Palette("dev2.pal");
  Refresh_Palette();
```

Buka mode graphic, instal palette dan aktifkan palette.

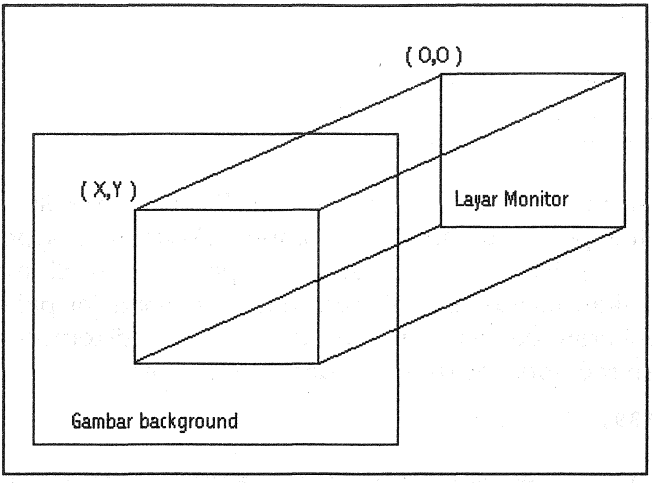
```
  VESA_ClearDouble(Buffer,0);
  LoadParallaxScroll("bk1");
```

Hapus virtual screen Buffer dan baca file gambar BK1 (berupa dua buah file yaitu BK1.SCR dan BK1.ATR)

```
  VESA_ClearDouble(Buffer,0);
  while (!finish)
  {
    VESA_ClearDouble(Buffer,0);
    PutParallaxScroll(x,y,Buffer,Temp);
```

Tampilkan parallax scrolling pada virtual screen Buffer dan dengan menggunakan Temp sebagai tempat kosong pembukaan gambar yang dikompres.

Adapun variabel x,y menunjukkan koordinat kiri atas dari back yang akan ditampilkan pada koordinat (0,0) dari virtual screen. Untuk lebih jelasnya, perhatikan Gambar 6.3.



Gambar 6.3 Bagian dari Gambar Background yang Ditampilkan ke Layar Ditentukan oleh Variabel (X,Y)

```
VESA_MoveToScreen(Buffer);
    while (!kbhit())
    {
    }
    ch=getch();
    if (ch==72) y-=10;
    if (ch==80) y+=10;
    if (ch==75) x-=10;
    if (ch==77) x+=10;
```

Gerakkan kursor untuk menggerakkan variabel (X,Y) dari gambar background sehingga dihasilkan efek bahwa gambar background bergeser.

```
    if (x<1) x=1;
    if (y<1) y=1;
    if (x>ParallaxData.Length-640) x=ParallaxData.Length-640;
    if (y>ParallaxData.Width-400) y=ParallaxData.Width-400;
```

Batasi pergeseran gambar background sehingga tidak melebihi ukuran dari gambar background tersebut.

```
    if (ch==27) finish=1;
```

Keluar jika tombol ESC ditekan.

```
  }
  CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

Sekarang kita lihat struktur data yang digunakan untuk menyimpan background dalam file header SCROLL.H.

```
#ifndef _SCROLL_H_
#define _SCROLL_H_

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "try.h"

struct _ParallaxData
{
  int Length,Width,Page_X,Page_Y,Last_X,Last_Y;
  char Name[20];
  long Size;
};
```

Record ParallaxData berisikan:

```
Length   = Jumlah titik horizontal dari background.
Width    = Jumlah titik vertikal dari background.
Page_X   = Jumlah potongan gambar horizontal.
Page_Y   = Jumlah potongan gambar vertikal.
Last_X dan Last_Y masih belum dipergunakan.
Name     = Nama file dari background.
Size     = Jumlah memory yang diperlukan untuk meny-
           impan background.
```

```
char * ParallaxPage[10000];
char  ParallaxFlag[10000];
```

ParallaxPage adalah array satu dimensi yang digunakan untuk menyimpan potongan-potongan sprite.

ParallaxFlag adalah array satu dimensi yang digunakan untuk menyimpan informasi apakah pada potongan sprite tersebut akan dilakukan fungsi VESA_PutImage atau cukup dilakukan VESA_Block.

```
_ParallaxData ParallaxData;

int LoadParallaxScroll(char * name);
```

```
void PutParallaxScroll(int x,int y,char * Buff,char * Temp);
void FreeParallaxScroll();

#endif
```

Nah, sekarang lihat implementasi dari fungsi-fungsi parallax scrolling ini dalam file SCROLL.CPP.

```
#include "try.h"
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "scroll.h"
```

Fungsi FreeParallaxScroll

Fungsi ini digunakan untuk menghapus seluruh potongan sprite dari memory.

```
void FreeParallaxScroll()
{
  int a;
  for (a=0;a<10000;a++)
  {
    if (ParallaxPage[a]!=NULL)
    {
      free(ParallaxPage[a]);
      ParallaxPage[a]=NULL;
    }
  }
```

Lakukan loop untuk menghapus seluruh isi array pada ParallaxPage.

```
  ParallaxData.Size=0;
```

Set harga ParallaxData.Size ke-0 yang menunjukkan belum ada data.

```
}
```

Fungsi LoadParallaxScroll

Fungsi ini digunakan untuk membaca file background untuk parallax scroll.

```
int LoadParallaxScroll(char * name)
{
```

Parameter yang dikirimkan ialah nama file tanpa ekstensi.

```
FILE * Fil;
FILE * Fildat;
int a,b;
int size;
char sFil[15];
char sFildat[15];
strcpy(sFil,name);
strcpy(sFildat,name);
strcat(sFil,".SCR");
strcat(sFildat,".ATR");
Fil=fopen(sFil,"rb");
```

Buka file dengan ekstensi SCR yang menampung potongan-potongan sprite yang telah dikompres.

```
if (Fil!=NULL)
{
Fildat=fopen(sFildat,"r");
```

Buka file dengan ekstensi ATR yang menampung informasi dari gambar background dan juga ukuran dari setiap potongan sprite.

```
for (a=0;a<10000;a++)
{
  ParallaxPage[a]=NULL;
}
fscanf(Fildat,"%d ",&ParallaxData.Length);
fscanf(Fildat,"%d ",&ParallaxData.Width);
fscanf(Fildat,"%d ",&ParallaxData.Page_X);
fscanf(Fildat,"%d ",&ParallaxData.Page_Y);
fscanf(Fildat,"%d ",&ParallaxData.Last_X);
fscanf(Fildat,"%d ",&ParallaxData.Last_Y);
```

Baca informasi dari gambar background.

```
a=1;
ParallaxData.Size=0;
while(!feof(Fil))
{
  fscanf(Fildat,"%d ",&size);
  ParallaxData.Size+=size;
  ParallaxPage[a]=(char *)malloc(size);
  ParallaxFlag[a]=1;
  fread(ParallaxPage[a],size,1,Fil);
  a++;
}
```

Lakukan loop untuk membaca seluruh potongan sprite dan masukkan ke dalam array ParallaxPage. Ukuran setiap potongan sprite ini dapat diketahui dari file dengan ekstensi ATR.

```
    fclose(Fil);
    fclose(Fildat);
    return 1;
    }
    return 0;
}
```

Fungsi PutParallaxScroll

Fungsi ini digunakan untuk menampilkan background parallax scrolling dengan koordinat kiri atas (X,Y) pada virtual screen.

```
void PutParallaxScroll(int X,int Y,char * Buff,char * Temp)
{
   int Al_X,Al_Y;
   int Xs,Ys;
   int Xr,Yr;
   int Pic_No;
   int Size;
```

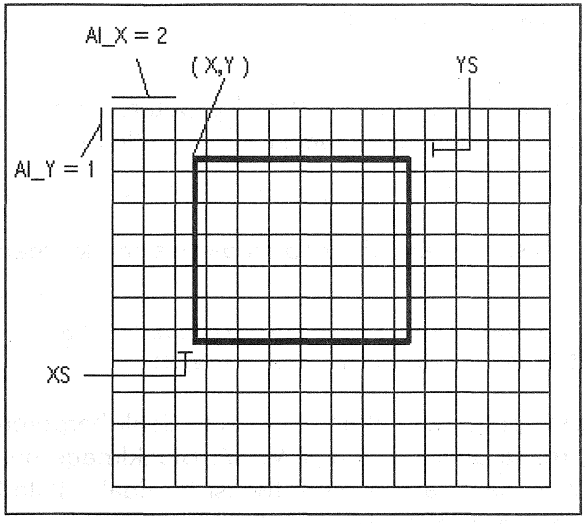
Parameter yang dikirimkan ialah (X,Y) yaitu koordinat offset dari background yang akan ditampilkan pertama kali. Buff yaitu virtual screen tempat background akan ditampilkan dan Temp yaitu memory kosong tempat hasil kompres akan diletakkan.

```
   Al_X=(X/80)+1;
   Al_Y=(Y/50)+1;
```

Hitung nomor potongan sprite kiri atas yang akan ditampilkan. Caranya ialah dengan membagi koordinat (X,Y) dengan ukuran dari setiap potongan sprite yaitu (80 x 50). Perhatikan Gambar 6.4. Setiap kotak kecil menunjukkan sebuah potongan sprite. Kotak bergaris tebal menunjukkan bagian dari background yang akan ditampilkan pada virtual screen.

```
   Xs=(-(X % 80));
   Ys=(-(Y % 50));
   Xr=Xs;Yr=Ys;
```

XS dan YS menunjukkan jarak antara pinggiran potongan sprite terdekat dengan koordinat (X,Y) dalam satuan titik.



Gambar 6.4 Perhitungan Potongan Sprite Kiri Atas yang Akan Diletakkan

```
do
{
  do
  {
    if (Al_Y>0&&Al_X>0)
    {
    Pic_No=((Al_Y-1)*ParallaxData.Page_X)+Al_X;
```

Dengan Al_Y, Al_X dan jumlah potongan gambar horizontal, kita dapat mengetahui indeks array yang harus kita akses, yaitu dengan rumus di atas. Kenapa Al_Y harus dikurangi 1? Karena harga Al_Y minimum ialah 1 sehingga kita harus menguranginya dengan 1 dan didapat harga Al_Y minimum = 0.

Tentu para pembaca mulai bertanya-tanya, kenapa kok Al_X tidak dikurangi 1. Harga minimum dari Al_X ternyata juga 1, bukannya 0. Memang sebenarnya Al_X juga harus dikurangi 1 tetapi di sini tidak kita lakukan, sehingga array dengan indeks nomor 0 tidak dipakai sama sekali. Kenapa dilakukan hal ini? Karena pada fungsi LoadParallaxScroll, kita mulai melakukan peletakan potongan sprite pada cell indeks nomor 1, bukannya 0.

Untuk mengubahnya sebenarnya tidak sulit, cukup mengganti a=1 pada fungsi LoadParallaxScroll dengan a=0 dan mengganti rumus di atas dengan menambahkan Al_X-1. Untuk itu diserahkan kepada pembaca.

```
if (ParallaxFlag[Pic_No]==0)
{
```

Jika harga pada ParallaxFlag = 0 maka potongan sprite harus diletakkan dengan menggunakan fungsi VESA_PutImage.

```
RllUnCompress(ParallaxPage[Pic_No],Temp);
```

Buka potongan sprite yang dikompres pada array ke memory kosong Temp.

```
if (Xr>=0&&Yr>=0&&Xr<=MaxX-80&&Yr<=MaxY-
  50&&Al_X<ParallaxData.Page_X)
```

Jika potongan sprite yang akan diletakkan tidak berpotongan dengan batas layar, maka gunakan fungsi VESA_BlockImage untuk meletakkan sprite. Kenapa? Karena pada fungsi ini tidak dilakukan pembatasan layar sehingga jauh lebih cepat.

```
  VESA_BlockImage(Buff,Xr,Yr,Temp);

  else
  {
    VESA_ClipBlockImage(Buff,Xr,Yr,Temp);
  }
```

Potongan sprite menembus batas layar, gunakan fungsi VESA_ClipBlockImage.

```
} else
{
  VESA_Block(Buff,Xr,Yr,Xr+79,Yr+49,
  ParallaxFlag[Pic_No]);
}
```

Karena harga ParallaxFlag pada potongan sprite ini tidak sama dengan 0 maka gunakan metode kedua, yaitu isi daerah di bawah potongan sprite dengan menggunakan VESA_Block.

Warna yang digunakan ialah harga yang disimpan dalam array ParallaxFlag.

```
}
else
{
  VESA_Block(Buff,Xr,Yr,Xr+79,Yr+49,0);
```

Jika tidak ada potongan gambar yang mewakili daerah ini, maka tampilkan balok hitam. Hal ini dapat terjadi jika kita menggeser background terlalu jauh sehingga melebihi batas-batasnya.

```
        }



        Xr+=80;
        Al_X++;
```

Lompat ke koordinat selanjutnya dan ke indeks horizontal selanjutnya dengan menambah Al_X.

```
      }
      while (Xr<=MaxX-1&&Al_X<=ParallaxData.Page_X);
```

Loop hingga seluruh potongan gambar horizontal telah ditampilkan.

```
      Xr=Xs;
      Al_X=(X/80)+1;
      Al_Y++;
      Yr+=50;
```

Lompat ke koordinat selanjutnya dan ke indeks vertikal selanjutnya dengan menambah Al_Y.

```
    }
    while (Yr<=MaxY-1&&Al_Y<=ParallaxData.Page_Y);
```

Loop hingga seluruh potongan gambar vertikal telah ditampilkan.

```
}
```

Berikutnya akan kita lihat fungsi-fungsi pendukung dari parallax scroll.

Fungsi VESA_ClipBlockImage

Fungsi ini sama persis dengan fungsi VESA_PutImage, hanya saja di sini warna 0 tetap digambar pada virtual screen. Tidak seperti pada fungsi VESA_PutImage di mana warna 0 tidak ditampilkan pada virtual screen.

```
VESA_ClipBlockImage          PROC        NEAR

        LOCAL    XWStart:DWord,XRStart:DWord,YWStart:DWord,
                 YRStart:DWord,XRead:DWORD,Widths :DWord,YRead
```

```
            :DWord,Lengths:DWord

    push    ebp
    mov     ebp,esp
    sub     esp, 64
    ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
            BUFFER:DWORD

    mov     esi, [BUFFER]
    mov     edi, [PICBUFFER]

    Xor     eax, eax
    lodsw
    mov     ecx, eax
    Mov     [Lengths],eax
    Mov     [XRead],eax
    lodsw
    Mov     [Widths],eax
    Mov     [YRead],eax
    Mov     [YRStart],0

    Mov     eax,[X1]
    cmp     eax,S_L
    jg      @ClbPI_Vin
    Add     eax,[Lengths]
    Sub     eax,1
    Cmp     eax,0
    jl      @ClbPI_Vin
    Mov     eax,[Y1]
    cmp     eax,S_W
    jg      @ClbPI_Vin
    Add     eax,[Widths]
    Sub     eax,1
    Cmp     eax,0
    jl      @ClbPI_Vin
```

Batalkan penggambaran sprite jika sprite berada di luar layar.

```
    Mov     eax,[Y1]
    Mov     [YWStart],eax
    Mov     [XRStart],0
    Mov     eax,[X1]
    Mov     [XWStart],eax
    cmp     eax,0
  Jge @ClbVESX1_Not_Negatives
    Neg     eax
    Mov     [XRStart],eax
```

```
    Mov     [XWStart],0
    Sub     [XRead],eax
@ClbVESX1_Not_Negatives:
    Mov     eax,[X1]
    Add     eax,[Lengths]
    Dec     eax
    Cmp     eax,[S_L]
    Jle     @ClbVESX1_No_Right_Crops
    Sub     eax,[S_L]
    Sub     [XRead],eax
@ClbVESX1_No_Right_Crops:
    Mov     eax,[Y1]
    Cmp     eax,0
    Jge     @ClbVESNo_Top_Crops
    Neg     eax
    Mov     [YWStart],0
    Mov     [YRStart],eax
    Sub     [YRead],eax
@ClbVESNo_Top_Crops:
    Mov     eax,[Y1]
    Add     eax,[Widths]
    Dec     eax
    Cmp     eax,[S_W]
    Jle     @ClbVESNo_Bot_Crops
    Sub     eax,[S_W]
    Sub     [YRead],eax
```

Potong sprite terhadap batas-batas layar.

```
@ClbVESNo_Bot_Crops:
    Add     edi,[XWStart]
    Mov     eax,[YWStart]
    mul     [A_SL]
    Add     edi,eax
```

Hitung offset memory dari titik pertama sprite pada virtual screen.

```
    Mov     eax,[YRStart]
    Mul     [Lengths]
    Add     esi,eax
    Add     esi,[XRStart]
```

Hitung offset sprite dari titik pertama sprite.

```
    Mov     ecx, [YRead]
```

Sediakan loop untuk lebar sprite.

```
@ClbVESvLoop1:
  Push ecx
  Push edi
  Push esi
  Mov ecx, [XRead]
```

Sediakan loop untuk panjang sprite.

```
  shr ecx,1
  jnc @Clb1
  movsb
```

Geser panjang sprite ke kanan 1 bit. Jika bit yang digeser tadi = 1 maka lakukan transfer data 1 byte.

Karena telah digeser satu kali ke kanan maka counter ecx sekarang dalam satuan 2 byte.

```
@Clb1:
  shr  ecx,1
  jnc @Clb2
  movsw
```

Geser ecx sekali lagi ke kanan. Jika bit yang digeser = 1 maka lakukan transfer data 2 byte.

Karena telah digeser satu kali lagi ke kanan maka counter ecx sekarang dalam satuan 4 byte (32 bit).

```
@Clb2:
  rep   movsd
```

Lakukan transfer data 32 bit sebanyak ecx.

```
  Pop esi
  Pop edi
  Add esi,[Lengths]
  Add edi,[A_SL]
  Pop ecx
```

Lompat ke koordinat Y selanjutnya.

```
  Loop @ClbVESvLoop1

@ClbPI_Vin :
  mov esp,ebp
  pop ebp
  ret
VESA_ClipBlockImage                ENDP
```

Fungsi VESA_BlockImage

Fungsi ini sama persis dengan fungsi VESA_ClipBlockImage hanya saja bedanya tidak dilakukannya pemotongan terhadap sprite. Perbedaan yang kedua ialah ukuran horizontal dari sprite selalu 80 byte sehingga dapat dilakukannya 32 bit data transfer.

```
VESA_BlockImage         PROC        NEAR

        push    ebp
        mov     ebp,esp
        ARG     PICBUFFER:DWORD, X1:DWORD, Y1:DWORD,
                BUFFER:DWORD

        mov     esi, [BUFFER]
        mov     edi, [PICBUFFER]

        xor     eax,eax
        lodsw
        mov     ebx,eax
        lodsw
        mov     ecx,eax
```

Baca panjang dan lebar sprite. Sediakan loop vertikal sebesar lebar sprite.

```
        mov     eax,[Y1]
        mul     [A_SL]
        add     edi,eax
        add     edi,[X1]
```

Hitung offset dari titik pertama sprite pada virtual screen.

```
@Bl1:
        mov     edx,ecx
        mov     ecx,20
```

Sediakan loop horizontal sebesar 20 x 32 bit.

```
        push    edi
        rep     movsd
```

Lakukan 32 bit data transfer sebesar 80 byte.

```
        mov     ecx,edx
        pop     edi
        add     edi,[A_SL]
        loop    @Bl1
```

Ulang hingga seluruh titik telah ditampilkan.

```
        mov     esp,ebp
        pop     ebp
        ret
VESA_BlockImage                 ENDP
```

Fungsi RllUnCompress

Fungsi ini digunakan untuk membuka sprite yang dikompres. Adapun cara kerjanya menggunakan metoda RLL yang telah kita bahas pada awal bab ini.

```
RllUnCompress                   PROC    NEAR

        push    ebp
        mov     ebp,esp
        ARG     SOURCE:DWORD, DEST:DWORD
```

Parameter yang dikirimkan ialah SOURCE yaitu alamat sprite, dan DEST yaitu memory kosong tempat hasil pembukaan kompres.

```
        Mov     edi,Dest
        Mov     esi,Source
        Xor     ebx,ebx
        Cld

@r1:
```

Arahkan edi dan esi ke Dest dan Source.

```
        Xor     ecx,ecx
        Lodsb
        Or      al,al
        Jz      @r2
```

Ambil data pertama, test apakah data tersebut berisi kode 0.

```
        Mov     cl,al
        Add     bx,cx
        Lodsb
        Rep     Stosb
```

Data tersebut bukan 0 maka paket ini pasti merupakan paket jenis 2. Untuk itu masukkan harga pada byte pertama tadi ke dalam cx yang menunjukkan jumlah perulangan yang akan dilakukan. Kemudian ambil warna titik yang sebenarnya pada byte berikutnya. Lakukan pengisian berulang sejumlah cx kali ke dalam alamat Dest.

```
        Loop    @r1
```

Ulang lagi ke loop untuk memeriksa data berikutnya.

```
@r2:
        Lodsb
        Or      al,al
```

Data tersebut merupakan paket jenis 1. Maka dari itu ambil byte berikutnya yang menunjukkan jumlah titik yang disimpan dalam paket ini. Test dahulu apakah jumlah tersebut 0, jika 0 maka data telah habis karena tanda 0 yang diikuti dengan 0 lagi adalah tanda henti dari data.

```
        Jz      @r3
        Mov     cl,al
        Add     bx,cx
        Rep     Movsb
        Loop    @r1
```

Data belum habis. Isikan jumlah titik yang akan dipindahkan ke dalam cx. Kemudian lakukan pemindahan data sebanyak cx kali. Setelah selesai, loop ke awal lagi untuk memeriksa data berikutnya.

```
@r3:
        mov     esp,ebp
        pop     ebp
        ret
RllUnCompress           ENDP
```

6.2 Fungsi Penampilan Tulisan pada Mode Graphic 8 Bit

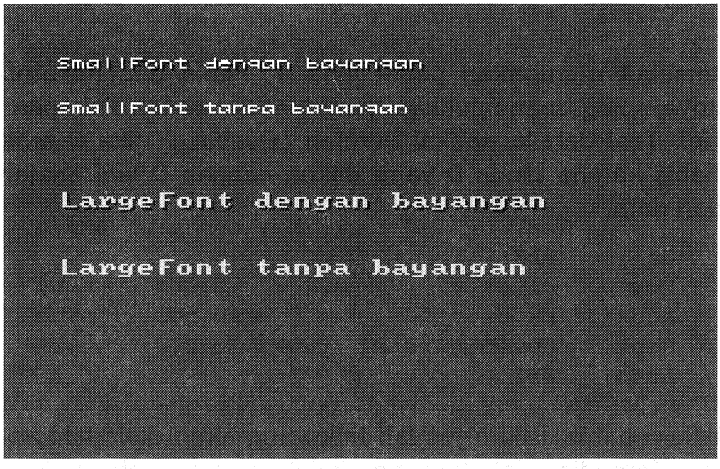
Materi terakhir yang akan kita bahas mengenai mode graphic 8 bit ialah penampilan tulisan ke layar. Library yang sama fungsinya dengan fungsi ini pada pembahasan kita terdahulu ialah VGAFONT.TPU.

Library yang akan kita buat ialah VGAFONT.CPP dengan file header VGAFONT.H.

Pembuatan library ini dilakukan dengan menganggap setiap karakter (huruf, angka dan tanda baca) sebagai sebuah sprite. Untuk itu disediakan sebuah array yang akan digunakan untuk menyimpan sprite dari masing-masing karakter. Sehingga pada saat kita hendak

menampilkan sebuah teks, dilakukan peletakan sprite pada setiap karakter dari teks tersebut.

Untuk lebih jelasnya, jalankan program TEKS.EXE. Perhatikan Gambar 6.5.



Gambar 6.5 Mencoba Fungsi-fungsi VGAFONT.CPP

Fungsi VGAFONT.CPP ini mendukung dua buah mode penampilan tulisan, yaitu dengan menggunakan bayangan dan tanpa menggunakan bayangan. Penampilan tulisan dengan bayangan akan menimbulkan efek gelap di sekitar tulisan yang ditampilkan sehingga memperjelas tulisan berwarna terang yang berada di atas background terang.

Bagaimana cara membuat efek ini? Sederhana, cukup dilakukan penampilan tulisan dua kali. Di mana pada penampilan pertama, tulisan ditampilkan agak bergeser dari posisi penampilan kedua dan menggunakan warna gelap. Sehingga setelah dilakukan penampilan kedua di atas penampilan pertama, pinggiran teks pada penampilan pertama masih kelihatan dan inilah yang menimbulkan efek gelap di sekitar teks.

Yuk, kita lihat unit VGAFONT.CPP beserta header filenya.

Ini adalah header file VGAFONT.H.

```
#ifndef _VGAFONT_H_
#define _VGAFONT_H_

typedef char * fontCell[256];
```

fontCell digunakan untuk menampung sprite dari setiap karakter. Penyimpanannya diurutkan sesuai dengan urutan ASCII dari karakter yang disimpan, sehingga nantinya pada saat penampilan dari teks, kita cukup mengirimkan nomor ASCII dari karakter yang hendak kita tampilkan sebagai indeks array ini.

```
void     LoadFont(fontCell,char *,int);
void          VESA_PutString(char   *,fontCell,long,long,long,char,char
*,long);

#endif
```

Sekarang kita lihat file VGAFONT.CPP.

```
#include "try.h"
#include "vgafont.h"
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Fungsi LoadFont

Fungsi ini digunakan untuk membaca sebuah file font. Dengan ini, kita dapat menggunakan font yang berbeda-beda dalam satu program.

```
void     LoadFont(fontCell usedFnt,char * strig,int size)
{
```

Parameter yang dikirimkan ialah usedFnt yaitu array yang akan digunakan untuk menyimpan gambar font, strig yaitu nama file tempat font disimpan dan size yang merupakan besar memory yang diperlukan untuk menyimpan sebuah sprite dari karakter. Jika sprite dari sebuah karakter berukuran 5 x 5 maka jumlah memory yang dibutuhkan ialah 5 x 5 + 4 = 29 Bytes (ingat pada pembahasan VESA_GetImage).

Seiring dengan buku ini, disediakan dua buah file font yaitu 5x5.FNT dan 8x8.FNT.

```
        int a;
        FILE * fntFile;
        fntFile=fopen(strig,"rb");
```

Buka file font.

```
for (a=0;a<126;a++)
{
   usedFnt[a]=(char *)malloc(size);
   fread(usedFnt[a],size,1,fntFile);
}
```

Baca 126 potongan sprite yang mewakili 126 karakter dan masukkan ke dalam array esedFnt. Kenapa hanya 126? Jumlah ASCII memang 256 tapi nomor 127 ke atas jarang sekali dipergunakan, sehingga tidak kita sediakan sprite untuk karakter tersebut. Tetapi untuk file font yang mendukung 256 font penuh, dapat juga disimpan dalam usedFnt karena struktur data ini memiliki kemampuan menyimpan 256 sprite.

```
}
```

Fungsi VESA_PutString

Fungsi ini digunakan untuk menampilkan string pada virtual screen. Dengan fungsi ini, kita dapat menentukan jenis font yang kita gunakan, letak tulisan, warna tulisan, jarak spasi antara karakter, tulisan yang akan ditampilkan, dan digunakannya bayangan atau tidak.

```
void VESA_PutString(char * back,fontCell usedFnt,long
x,long y,long
                    fsize,char col,char * str,long shad)
{
```

Parameter yang dikirimkan ialah:

```
Back     = yaitu virtual screen di mana teks akan di-
           tampilkan.
UsedFnt  = ialah nama array font yang akan digunakan.
X,Y      = ialah letak teks.
Fsize    = ialah ukuran spasi antara karakter.
Col      = ialah warna teks yang akan ditampilkan.
Shad     = jika diisi 1 maka teks akan ditampilkan
           dengan bayangan.
```

```
  int a,b;
  for (a=0;a<strlen(str);a++)
  {
```

Lakukan loop untuk mengakses seluruh karakter dalam teks.

```
    b=*(str+a);
```

Ambil karakter yang sedang diproses sesuai dengan counter dari loop.

```
if ((b>0)&&(b<126))
{
  if (shad)
     VESA_Put1Col(back,x+a*fsize+1,y+1,usedFnt[b],0);
  VESA_Put1Col(back,x+a*fsize,y,usedFnt[b],col);
```

Jika karakter berada antara ASCII 0 hingga 125 maka tampilkan karakter tersebut dengan menggunakan VESA_Put1Col sehingga warna karakter dapat kita atur. Gunakan fsize sebagai jarak horizontal antara tiap karakter. Jika shad berharga 1 maka tampilkan bayangan dari teks yang diciptakan dengan cara peletakan teks yang sama dengan koordinat yang bergeser 1 ke kanan dan ke bawah dan dengan warna 0 = hitam.

Karakter tersebut digunakan sebagai indeks array usedFnt untuk mengambil sprite dari karakter tersebut.

```
    }
  }
}
```

Sekarang kita akan membahas program TEKS.CPP di atas.

```
/* ------------------------------------------------------
Nama File   : TEXT.CPP
Definisi    : Program mencoba fungsi font
Programmer  : Agustinus Nalwan
------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "try.h"
#include "shape.h"
#include "shape.cpp"
#include "vgafont.h"
#include "vgafont.cpp"
```

Definisikan penggunaan library VGAFONT.CPP dan file headernya.

```
char ch;
char * Buffer;
fontCell SmallFont,LargeFont;
```

Deklarasikan dua buah array fontCell untuk menyimpan dua macam font.

```
int main()
{
  Buffer=(char *)malloc(MEM400);
```

Alokasikan memory untuk virtual screen Buffer.

```
  OpenVGA256(MOD400);
```

Buka mode graphic.

```
  VESA_ClearDouble(Buffer,8);
  LoadFont(SmallFont,"5x5.fnt",29);
  LoadFont(LargeFont,"8x8.fnt",68);
```

Baca dua buah file font dan masukkan ke dalam SmallFont dan LargeFont. Ukuran sprite LargeFont ialah 8 x 8 + 4 = 68 Bytes.

```
  VESA_PutString(Buffer,SmallFont,50,40,6,15,"SmallFont dengan
          bayangan",1);
  VESA_PutString(Buffer,SmallFont,50,60,6,15,"SmallFont tanpa
          bayangan",0);
  VESA_PutString(Buffer,LargeFont,50,100,8,14,"LargeFont dengan
          bayangan",1);
  VESA_PutString(Buffer,LargeFont,50,130,8,14,"LargeFont tanpa
          bayangan",0);
```

Tampilkan tulisan dengan menggunakan mode bayangan dan mode normal dengan memainkan parameter shad.

```
  VESA_MoveToScreen(Buffer);
```

Tampilkan virtual screen ke layar monitor.

```
  ch=getch();
  CloseVGA256();
```

Tutup mode graphic.

```
  return 1;
}
```

Dengan ini selesailah pembahasan perakitan graphics library 8 bit. Dengan ini juga, selesailah pembahasan kita pada edisi kali ini. Oppppss lupa, kita masih belum membahas tentang programming di 16, 24 dan 32 bit kan? Ok... kita teruskan pembahasan mode-mode graphic ini pada bab berikutnya.

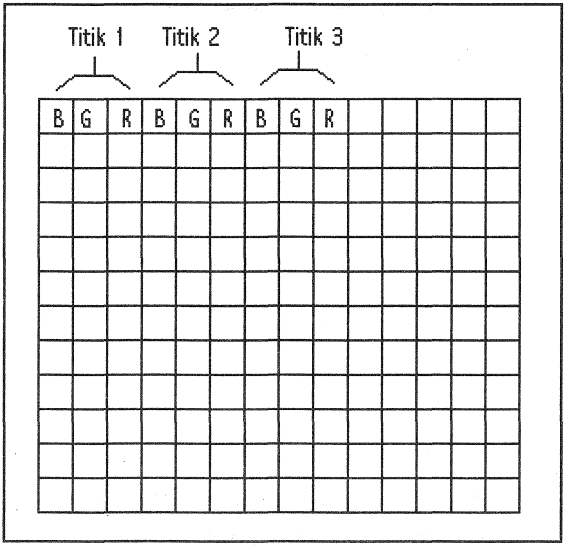
BAB 7

PEMROGRAMAN SUPER VGA 16, 24, 32 BIT

Seperti yang telah kita bahas pada Bab 4. Pemrograman Super VGA dengan mode di atas 8 bit harus dilakukan tanpa tabel warna. Sehingga setiap titik pada layar memiliki intensitas RGB sendiri. Dengan ini banyak keuntungan yang didapat yaitu kita tidak perlu pusing-pusing mencari warna pengganti dalam tabel warna jika kita hendak mengganti intensitas warna sebuah titik.

Nah, jarang card Super VGA yang mendukung semua mode resolusi walaupun keperluan memory untuk mode resolusi tersebut telah dicukupi. Misalnya saja mode graphic 320 x 200 24 bit. Tidak semua card mendukung mode graphic ini. Tetapi, untuk menghindari hal ini kita akan melakukan pembahasan pada mode graphic 640 x 480 24 bit yang kebanyakan didukung oleh semua card Super VGA. Lantas bagaimana mode graphic yang lain? Jika kita telah mengetahui bagaimana cara memprogram mode graphic 24 bit, pemrograman mode 16 dan 32 bit bukanlah menjadi masalah karena tekniknya sama. Yang berbeda hanyalah jumlah byte yang diperlukan per titiknya, dan tentu saja nomor register yang harus dikirimkan untuk membuka mode graphic tersebut. Hal ini telah kita bahas pada Bab 3.

Untuk itu kita akan merakit sebuah graphic library sederhana (tidak selengkap library 8 bit yang telah kita bahas) yang memanfaatkan mode graphic 24 bit. Tetapi jangan khawatir, beberapa fungsi yang akan kita bahas akan membuat Anda terkejut akan kehebatan dan kelebihan mode graphic 24 bit seperti efek gamma, efek multi level transparansi, pencahayaan yang mendukung seluruh kemungkinan warna tanpa perlunya tabel dan lain sebagainya.



Gambar 7.1 Model Penyimpanan Titik dalam Virtual Screen Mode 4 Bit

Pada mode 24 bit, karena setiap titik memiliki intensitas R, G dan B yang masing-masingnya 8 bit, maka untuk menyimpan satu buah titik dalam virtual screen diperlukan 3 byte. Adapun urutan penyimpanan ini dapat kita lihat pada Gambar 7.1. Masing-masing kotak menunjukkan 1 byte memory. Dari grup 3 byte untuk setiap titik, byte pertama menyimpan intensitas B, kedua menyimpan intensitas G dan ketiga menyimpan intensitas R.

Nah, karena untuk meletakkan satu buah titik, kita perlu memberitahukan intensitas R, G dan B-nya maka fungsi PutPixel kita nantinya akan memiliki tiga buah parameter warna.

7.1 Graphic Library 24BIT.ASM

Sama seperti pada perakitan graphic library 8 bit, di sini digunakan sebuah external assembler yaitu 24BIT.ASM yang isinya hanyalah

fungsi-fungsi standar dari mode graphic 24 bit. Kebanyakan fungsi-fungsi geometri dan penampilan sprite akan dirakit dengan C untuk mempermudah dan mempersingkat pembahasan. Untuk menghubungkan external assembler file ini dengan C, kita akan menggunakan sebuah file header yang bernama 24BIT.H.

Sama halnya dengan graphic library 8 bit terdahulu yang banyak dibantu oleh unit-unit lain seperti VGAPAL.CPP, SHAPE.CPP dan VGAFONT.CPP. Di sini kita juga akan menggunakan beberapa unit bantu yaitu IMAGE.CPP dan SHAPE.CPP (file SHAPE.CPP ini lain dengan file SHAPE.CPP pada graphic library 8 bit).

Para pembaca tentu saja sudah tidak tahan ingin mencoba mode graphic ini khan? Ayo, coba jalankan program TBLOCK.EXE. Pada layar akan ditampilkan balok-balok kecil yang warnanya terus berubah mulai warna biru hingga merah. Hal ini tidak dapat kita ciptakan pada mode graphic 256 warna karena jumlah warna yang tidak memadai. Tapi pada mode graphic 24 bit, tidak lagi.

Yuk, kita lihat listing programnya.

```
/* ------------------------------------------------------------
Nama            : TBLOCK.CPP
Definisi        : Program untuk mencoba mode graphic 24 bit
Programer       : Agustinus Nalwan
------------------------------------------------------------ */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
```

Definisikan header file 24BIT.H dan library SHAPE.CPP beserta SHAPE.H. Ketiga file ini akan kita bahas setelah kita membahas program ini.

```
char * buffer;
char ch;

void test()
{
  long x,y,ctr;
  long r=0,g=0,b=0;
```

Set warna r = 0, g = 0 dan b = 0.

```
for (x=0;x<128;x++)
{
  for (y=0;y<96;y++)
  {
```

Lakukan loop sebesar 128 horizontal dan 96 vertikal untuk menciptakan kotak kecil sejumlah 128 x 96 buah.

```
Block(buffer,x*5,y*5,x*5+4,y*5+4,r,g,b);
```

Tampilkan kotak pada virtual screen dengan warna r,g,b.

```
b+=2;
g+=2;
```

Tambah intensitas b dan g untuk penggambaran horizontal berikutnya.

Penambahan kedua intensitas ini menyebabkan kotak cenderung memiliki unsur biru laut (gabungan antara biru dan hijau) yang semakin bertambah.

Efek ini dapat kita lihat pada tampilan di mana semakin ke kanan, unsur biru laut semakin terlihat.

```
}
b=0;
g=0;
r+=2;
```

Set kembali b dan g ke 0 dan tambahkan r untuk koordinat vertikal berikutnya. Dengan ini, semakin ke bawah, unsur merah akan semakin kelihatan.

```
  }
  MoveToScreen(buffer);
  ch=getch();
```

Tampilkan virtual screen ke layar.

```
}

int main()
{
  OpenVGA24BIT(MOD640);
```

Buka mode graphic 640 x 480 24 bit. MOD640 ini adalah konstanta yang dideklarasikan pada file header 24BIT.H.

```
MaxX=639;MaxY=479;
```

Selalu ingat untuk mengisi dua variabel ini sesuai dengan ukuran layar yang sedang aktif.

```
buffer=(char *)malloc(MEM640);
```

Alokasikan memory untuk virtual screen buffer sebesar MEM640 yang merupakan konstanta pada file header 24BIT.H. Ukuran dari MEM640 ini ialah 640 x 480 x 3 = 921.600 Bytes. Besar sekali kan? Ini masih belum apa-apa. Graphics library ini mendukung resolusi-resolusi pada Tabel 7.1.

Tabel 7.1 Mode Graphic yang Didukung Graphic Library 24 bit

Resolusi	Jumlah bit per titik	Jumlah warna	Keperluan memory
320 x 200	24	16 Juta	19.200
640 x 480	24	16 Juta	921.600
800 x 600	24	16 Juta	1.440.000
1024 x 768	24	16 Juta	2.359.296
1280 x 1024	24	16 Juta	3.932.160

```
test();
```

Panggil subroutine untuk menampilkan kotak ke layar.

```
CloseVGA24BIT();
```

Tutup mode graphic.

```
  return 1;
}
```

Mode resolusi 1280 x 1024 membutuhkan memory hingga 4 Mega-Bytes. Bayangkan, berapa pelan jika kita harus melakukan penggantian virtual screen secara manual dengan transfer data. Untuk itu mode-mode graphic ini jarang sekali digunakan untuk pembuatan game yang membutuhkan kecepatan tinggi. Kecuali jika dibutuhkannya spesial graphic card yang membantu proses-proses graphic seperti yang sekarang sedang ramainya dibicarakan, 3D card.

Ok, sekarang kita akan melihat file header 24BIT.H yang digunakan sebagai penghubung antara external assembler 24BIT.ASM dengan C.

```
#ifndef _24BIT_H_
#define _24BIT_H_

#define SCREEN (char *)(0x0A0000)
#define MEM1280 3932160
#define MEM1024 2359296
#define MEM800  1440000
#define MEM640  921600
#define MEM320  192000
```

Konstanta di atas digunakan sebagai petunjuk jumlah memory yang diperlukan oleh setiap mode graphic.

```
#define MOD1280 0x11b
#define MOD1024 0x118
#define MOD800  0x115
#define MOD640  0x112
#define MOD320  0x10f
```

Konstanta di atas digunakan sebagai petunjuk angka yang harus dimasukkan ke register eax pada saat membuka mode graphic untuk setiap mode graphic.

```
int MaxX,MaxY;
```

Dua buah variabel yang digunakan sebagai patokan bagi seluruh fungsi graphic yang menunjukkan koordinat maksimum horizontal dan vertikal.

```
extern "C"
{
  extern void OpenVGA24BIT(long mode);
  extern void CloseVGA24BIT();
  extern void MoveDouble(char *source, char *dest);
  extern void ClearDouble(char *source,char Col);
  extern void MoveToScreen(char *PICBUFFER);
  extern void PutPixel(char *BUFFER,long x,long y,long cx);
}
```

Fungsi-fungsi di atas ialah fungsi yang disediakan dalam file 24BIT.ASM. Terlihat sangat sedikit karena kebanyakan fungsi graphic akan dirakit dalam file IMAGE.CPP menggunakan C.

```
#endif
```

Selanjutnya kita lihat file 24BIT.ASM yang berisi fungsi-fungsi yang kita pergunakan pada program di atas.

```
; ------------------------------------------
; Nama        : 24BIT.ASM
; Definisi    :
; VGA 24-BIT Graphics Library
; Resolution Support  :
;             320 x  200
;             640 x  480
;             800 x  600
;            1024 x  768
;            1280 x 1024
; Programmer : Agustinus Nalwan
; ------------------------------------------

.486P
.MODEL FLAT
```

Karena digunakan mode pemrograman 32 bit maka deklarasikan memory sebagai FLAT (tanpa segment-offset).

```
.DATA
        A_SL    DD      ?
        A_SW    DD      ?
        W_Scr   DD      ?
        W_Loop  DD      ?
        W_Acc   DD      ?
        BUFSIZE DD      ?
```

Definisikan variabel yang sama persis seperti yang kita gunakan pada mode graphic 8 bit. BUFSIZE menunjukkan besar memory yang diperlukan untuk sebuah virtual screen dalam satuan 32 bit.

```
_TEXT SEGMENT DWORD PUBLIC 'CODE'
        ASSUME cs:_TEXT, ds:_DATA

public  OpenVGA24BIT
public  CloseVGA24BIT
public  MoveDouble
public  MoveToScreen
public  ClearDouble
public  PutPixel
```

7.2 Fungsi-fungsi Standar 24BIT.ASM

Fungsi-fungsi standar ini terdiri atas fungsi-fungsi yang digunakan untuk membuka, menutup mode graphic 24 bit, juga untuk memindahkan virtual screen ke virtual screen yang lain, menghapus sebuah virtual screen dan menampilkan virtual screen ke layar.

Fungsi MoveDouble

Fungsi ini digunakan untuk memindahkan virtual screen ke virtual screen yang lain.

```
MoveDouble      PROC NEAR
        push    ebp
        mov     ebp, esp
        ARG     SOURCE:DWORD, DEST:DWORD
```

Parameter yang dikirimkan ialah alamat virtual screen yang akan dipindahkan yaitu SOURCE dan alamat virtual screen yang dituju yaitu DEST.

```
        mov     esi, [SOURCE]
        mov     edi, [DEST]
        mov     ecx, [BUFSIZE]
        rep     movsd
```

Lakukan transfer data sebesar BUFSIZE.

```
        mov     esp, ebp
        pop     ebp
        ret

MoveDouble      ENDP
```

Fungsi ClearDouble

Fungsi ini digunakan untuk mengisi virtual screen dengan warna putih dengan berbagai macam intensitas. Karena parameter warna hanya satu yaitu COL sehingga ketiga intensitas R, G dan B akan diisikan dengan variabel ini. Kenapa kok tidak memisahkan parameter R, G dan B secara terpisah sehingga dimungkinkannya penghapusan layar dengan warna bebas? Fungsi ini hanya dipergunakan untuk menghapus layar dengan warna hitam saja, sehingga tidak ketiga unsur intensitas R, G dan B diisikan dengan 0, jadi cukup digunakan satu variabel saja. Selain itu kita juga dapat mengisikan layar dengan

warna putih terang yaitu dengan menggunakan parameter yang berharga 255 sehingga ketiga unsur R, G dan B diisi dengan 255. Jika kita hendak membuat penghapusan layar dengan sebuah warna, sebaiknya kita gunakan fungsi terpisah karena setiap unsur R, G dan B yang berbeda tentu membutuhkan 8 bit data transfer yang terpisah sehingga mengakibatkan lambatnya program. Karena kita menggabungkan ketiga unsur R, G dan B ini dalam satu parameter maka kita dapat melakukan transfer data 32 bit yang sangat cepat.

```
ClearDouble     PROC NEAR
        push    ebp
        mov     ebp,esp
        ARG     SOURCE:DWORD,COL:BYTE
        mov     edi, [SOURCE]
```

Arahkan edi ke virtual screen yang akan dihapus.

```
        mov     al,[COL]
        mov     ah,[COL]
        xor     edx,edx
        mov     dx,ax
        shl     eax,16
        add     eax,edx
```

Isi 4 byte dalam register eax dengan COL.

```
        mov     ecx,[BUFSIZE]
        rep     stosd
```

Lakukan transfer data sejumlah BUFSIZE untuk mengisi seluruh unsur warna dari setiap titik.

```
        mov     esp,ebp
        pop     ebp
        ret

ClearDouble     ENDP
```

Fungsi OpenVGA24BIT

Fungsi ini digunakan untuk membuka mode graphic 24 bit. Teknik pemrogramannya hampir sama dengan OpenVGA256 pada VGA.ASM.

```
OpenVGA24BIT PROC NEAR

        push    ebp
```

```
        mov     ebp,esp
        ARG     mode:DWORD

        mov     eax,4f02h
        mov     ebx,[mode]
        int     10h
```

Buka mode graphic sesuai dengan mode yang dikirimkan.

```
        mov     ebx,[mode]
        cmp     ebx,112h
        je      @mode112
        cmp     ebx,10fh
        je      @mode10f
        cmp     ebx,115h
        je      @mode115
        cmp     ebx,118h
        je      @mode118
        cmp     ebx,11bh
        je      @mode11b
        jmp     @finmode

    @mode115:
        mov     eax,115h
        int     10h
        mov     [A_SL],2400
        mov     [A_SW],600
        mov     [W_Scr],1440000
        mov     [BUFSIZE],360000
        mov     [W_Loop],21
        mov     [W_Acc],15936
        jmp     @finmode
```

Mode graphic yang dipilih ialah 800 x 600 24 bit.

A_SL = Jumlah byte horizontal = 2400. Kenapa? Karena setiap titik memiliki 3 byte yang digunakan untuk masing-masing unsur R, G dan B. Dan jumlah titik horizontal = 800 sehingga diperlukan 3 x 800 = 2400 byte setiap garis horizontal.

A_SW = Jumlah titik vertikal = 600.

W_Scr = Jumlah memory yang diperlukan untuk virtual screen dalam byte.

BUFSIZE = Jumlah memory yang diperlukan untuk virtual screen dalam 32 bit.

```
W_Loop   = Jumlah jendela-1 yang diperlukan.
W_Acc    = Sisa byte pada jendela terakhir dalam 32
           bit.
```

```
@mode118:
  mov     eax,118h
  int     10h
  mov     [A_SL],3072
  mov     [A_SW],768
  mov     [W_Scr],2359296
  mov     [BUFSIZE],589824
  mov     [W_Loop],36
  mov     [W_Acc],0
  jmp     @finmode
```

Mode graphic yang dipilih ialah 1024 x 768 24 bit.

```
@mode11b:
  mov     eax,11bh
  int     10h
  mov     [A_SL],3840
  mov     [A_SW],1024
  mov     [W_Scr],3932160
  mov     [BUFSIZE],983040
  mov     [W_Loop],60
  mov     [W_Acc],0
  jmp     @finmode
```

Mode graphic yang dipilih ialah 1280 x 1024 24 bit.

```
@mode112:
  mov     eax,112h
  int     10h
  mov     [A_SL],1920
  mov     [A_SW],480
  mov     [W_Scr],921600
  mov     [BUFSIZE],230400
  mov     [W_Loop],14
  mov     [W_Acc],1024
  jmp     @finmode
```

Mode graphic yang dipilih ialah 640 x 480 24 bit.

```
@mode10f:
  mov     [A_SL],960
  mov     [A_SW],200
  mov     [W_Scr],192000
  mov     [BUFSIZE],48000
```

```
        mov     [W_Loop],2
        mov     [W_Acc],15232
        jmp     @finmode
```

Mode graphic yang dipilih ialah 320 x 200 24 bit.

```
    @finmode:
        mov     esp,ebp
        pop     ebp
        ret

OpenVGA24BIT ENDP
```

Fungsi CloseVGA24BIT

Fungsi ini digunakan untuk menutup mode graphic 24 bit. Caranya ialah dengan memanggil interrupt DOS untuk menampilkan mode teks.

```
CloseVGA24BIT     PROC NEAR

        mov     ax,2
        int     10h
        ret

CloseVGA24BIT     ENDP
```

Fungsi MoveToScreen

Fungsi ini digunakan untuk menampilkan virtual screen ke layar monitor. Implementasi fungsi ini sama persis seperti fungsi VESA_MoveToScreen pada VGA.ASM yaitu dengan mengirimkan potongan gambar pada virtual screen satu per satu ke dalam jendela card Super VGA.

```
MoveToScreen PROC NEAR
        push    ebp
        mov     ebp,esp
        ARG     SOURCE:DWORD
```

Parameter yang dikirimkan ialah alamat virtual screen yang akan ditampilkan.

```
        mov     ecx,[W_Loop]
        mov     esi, [SOURCE]
```

Set counter jendela sejumlah W_Loop.

```
@rep:
  xor     ebx,ebx
  mov     edx,[W_Loop]
  sub     edx,ecx
  mov     eax,00004f05h
  int     10h
```

Aktifkan nomor jendela yang sedang diproses.

```
  push    ecx
  mov     edi, 0a0000h
  mov     ecx, 16384
  rep     movsd
  pop     ecx
```

Lakukan transfer data sejumlah 64 Kbytes dari virtual screen ke dalam alamat memory $A0000.

```
  loop    @rep
```

Loop hingga sampai pada jendela terakhir.

```
  mov     edi, 0a0000h
  mov     ecx, [W_Acc]
  cmp     ecx,0
  jz      @noaccess
```

Jika sisa byte pada jendela terakhir = 0 maka lompati proses berikut.

```
  mov     eax,00004f05h
  xor     ebx,ebx
  mov     edx,[W_Loop]
  int     10h
```

Aktifkan jendela terakhir.

```
  rep     movsd
```

Isikan sisa byte pada jendela terakhir.

```
@noaccess:
  mov     esp,ebp
  pop     ebp
  ret

MoveToScreen ENDP
```

Fungsi PutPixel

Fungsi ini adalah satu-satunya fungsi geometri yang dirakit dalam Assembler. Gunanya ialah untuk menampilkan sebuah titik pada virtual screen.

```
PutPixel        PROC NEAR

        push    ebp
        mov     ebp,esp
      ARG     BUFFER:DWORD, X:DWORD, Y:DWORD, CXC:DWORD
```

Parameter yang dikirimkan ialah BUFFER yaitu alamat virtual screen yang akan digambari. X dan Y ialah koordinat titik dan CXC ialah variabel 32 bit di mana 3 byte pertamanya berisi unsur R, G dan B yang akan ditampilkan sebagai warna titik.

Misalnya harga CXC = $00442213 maka unsur R = $44, G = $22 dan B = $13. Dengan ini, kita tidak perlu mengirimkan tiga buah parameter warna.

```
        mov     eax,[Y]
        mul     [A_SL]
        mov     ebx,[X]
        add     eax,ebx
        add     eax,ebx
        add     eax,ebx
```

Hitung offset memory dari titik. Kenapa dilakukan tiga kali penambahan dengan koordinat X? Karena sekarang setiap titik terdiri dari tiga byte, sehingga untuk mengakses koordinat yang bersangkutan kita harus mengalikan jumlah titik dengan jumlah byte per titik. Kenapa koordinat Y tidak dikalikan 3? Karena konstanta A_SL telah berisi 3 x panjang horizontal sehingga secara langsung perkalian Y dengan A_SL di atas telah termasuk perkalian dengan 3.

```
        mov     edi,[BUFFER]
        add     edi,eax
```

Tambahkan eax yang merupakan offset dari titik ke alamat virtual screen.

```
        mov     eax,[CXC]
```

Isikan CXC ke eax.

```
        stosb
        shr     eax,8
```

Isikan unsur B, kemudian geser CXC 8 bit ke kanan, sehingga sekarang 8 bit terkecil memegang unsur G.

```
        stosb
        shr     eax,8
```

Isikan unsur G kemudian geser CXC 8 bit ke kanan lagi, sehingga sekarang 8 bit terkecil memegang unsur R.

```
        stosb
```

Isikan unsur R.

```
        mov     esp,ebp
        pop     ebp
        ret
PutPixel        ENDP

        ENDS

END
```

Unit berikutnya yang juga kita gunakan pada program TBLOCK.EXE di atas ialah SHAPE.CPP. Unit ini dirancang untuk menyediakan fungsi-fungsi penggambaran bangun geometri sederhana seperti balok, persegi panjang, garis lurus horizontal dan vertikal maupun penggambaran window sederhana. Berikut ini adalah pembahasan dari unit tersebut.

```
#include "24bit.h"
#include "shape.h"
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Fungsi _PutPixel

Fungsi ini digunakan untuk menyederhanakan fungsi PutPixel pada 24BIT.ASM. Karena pemanggilan fungsi PutPixel pada 24BIT.ASM tersebut menggunakan variabel warna yang telah mewakili R, G dan B sehingga setiap kali kita melakukan PutPixel, kita harus melakukan perhitungan untuk menggabung ketiga unsur ini. Dengan fungsi ini, kita cukup mengirimkan unsur R, G dan B secara terpisah.

```
void _PutPixel(char * Buff,long X1,long Y1,long r,long
g,long b)
```

```
{
  long cx;
  cx=r*65536+g*256+b;
```

Gabung parameter R, G dan B ke dalam sebuah variabel dengan urutan, R pada byte ketiga, G pada byte kedua dan B pada byte pertama.

```
  PutPixel(Buff,X1,Y1,cx);
```

Panggil fungsi PutPixel dalam 24BIT.ASM dengan parameter cx yang merupakan gabungan antara unsur-unsur R, G dan B.

```
}
```

Fungsi Block

Fungsi ini digunakan untuk menggambar balok pada virtual screen. Fungsi ini kita pergunakan untuk menampilkan balok kecil-kecil pada virtual screen pada program TBLOCK.EXE di atas.

```
void Block(char * Buff,long x1,long y1,long x2,long y2,long
r,long g,long b)
{
```

Parameter yang dikirimkan ialah buff yaitu alamat virtual screen tempat balok akan digambar. (x1,y1)-(x2,y2) ialah koordinat kiri atas dan kanan bawah tempat balok akan digambar dan r, g, b merupakan intensitas warna dari balok yang akan digambar.

```
  int x,y;
  for (x=x1;x<=x2;x++)
  {
    for (y=y1;y<=y2;y++)
    {
```

Ciptakan loop untuk memproses seluruh titik dalam balok.

```
      _PutPixel(Buff,x,y,r,g,b);
```

Panggil fungsi _PutPixel untuk mengisi titik dalam balok yang sedang diproses dengan unsur warna r,g,b.

```
    }
  }
}
```

Nah, sekalian kita akan membahas beberapa fungsi yang lain yang tidak dipakai dalam program TBLOCK.EXE yang berada pada unit SHAPE.CPP ini.

Fungsi VLine

Fungsi ini digunakan untuk menampilkan garis vertikal pada virtual screen.

```
void VLine(char * Buff,long X1,long Y1,long Y2,long r,long
g,long b)
{
```

Parameter yang dikirimkan ialah Buff yaitu alamat virtual screen yang akan digambari. (X1,Y1) ialah koordinat titik pertama dari garis, dan Y2 ialah koordinat Y titik kedua dari garis. R, G dan B ialah intensitas warna dari garis yang akan ditampilkan.

```
  long a;
  if (Y1>Y2)
  {
    a=Y1;
    Y1=Y2;
    Y2=a;
  }
```

Tukar Y1 dengan Y2 jika Y1 di bawah Y2.

```
  if (Y2>=MaxY) Y2=MaxY;
  if (Y1<=0) Y1=0;
```

Batasi dengan garis pada sisi atas dan bawah layar.

```
  for (a=Y1;a<=Y2;a++)
  {
    _PutPixel(Buff,X1,a,r,g,b);
  }
```

Tampilkan titik-titik garis dari atas ke bawah dengan pemanggilan fungsi _PutPixel.

```
}
```

Fungsi HLine

Fungsi ini digunakan untuk menampilkan garis horizontal pada virtual screen.

```
void HLine(char * Buff,long X1,long X2,long Y1,long r,long
g,long b)
{
  long a;
  if (X1>X2)
  {
    a=X1;
    X1=X2;
    X2=a;
  }
  if (X2>=MaxX) X2=MaxX;
  if (X1<=0) X1=0;
  for (a=X1;a<=X2;a++)
  {
    _PutPixel(Buff,a,Y1,r,g,b);
  }
}
```

Fungsi Rect

Fungsi ini digunakan untuk menggambar persegi panjang pada virtual screen. Caranya ialah menggambar sisi-sisi dari persegi panjang secara terpisah dengan pemanggilan fungsi VLine dan HLine.

```
void Rect(char * Buff,long X1,long Y1,long X2,long Y2,long
r, long g,long b)
{
  VLine(Buff,X1,Y1,Y2,r,g,b);
  VLine(Buff,X2,Y1,Y2,r,g,b);
  HLine(Buff,X1,X2,Y1,r,g,b);
  HLine(Buff,X1,X2,Y2,r,g,b);
}
```

Fungsi DrawWindow

Fungsi ini digunakan untuk menggambar sebuah window pada virtual screen. Window di sini ialah kotak berbingkai yang biasa digunakan untuk alas dari menu.

```
void DrawWindow(char * Buff,long x1,long y1,long x2,long
y2,long style)
{
```

Parameter yang dikirimkan ialah Buff yaitu alamat memory virtual screen yang akan digambari. (x1,y1)-(x2,y2) ialah koordinat kiri atas dan kanan bawah dari window. Style yaitu nomor array WindowStyle yang digunakan.

```
Rect(Buff,x1,y1,x2,y2,WindowStyle[style].outframe.R,
    WindowStyle[style].outframe.G,WindowStyle[style].outframe.B);
```

Tampilkan bingkai terluar dengan susunan intensitas warna yang disimpan dalam WindowStyle[style].outframe.

```
Rect(Buff,x1+1,y1+1,x2-1,y2-1,WindowStyle[style].midframe.R,
    WindowStyle[style].midframe.G,WindowStyle[style].midframe.B);
```

Tampilkan bingkai tengah dengan susunan intensitas warna yang disimpan dalam WindowStyle[style].midframe.

```
Rect(Buff,x1+2,y1+2,x2-2,y2-2,WindowStyle[style].inframe.R,
    WindowStyle[style].inframe.G,WindowStyle[style].inframe.B);
```

Tampilkan bingkai terdalam dengan susunan intensitas warna yang disimpan dalam WindowStyle[style].inframe.

```
Rect(Buff,x1+3,y1+3,x2-3,y2-3,WindowStyle[style].bound.R,
    WindowStyle[style].bound.G,WindowStyle[style].bound.B);
```

Tampilkan pembatas terdalam dengan susunan warna dalam WindowStyle[style].bound.

```
Block(Buff,x1+4,y1+4,x2-4,y2-4,WindowStyle[style].bgr.R,
    WindowStyle[style].bgr.G,WindowStyle[style].bgr.B);
```

Terakhir ialah penampilan background dari window dengan warna dalam WindowStyle[style].bgr.

```
}
```

Array WindowStyle digunakan untuk menyimpan kombinasi warna dari bermacam-macam window. Satu cell dalam array WindowStyle menyimpan satu jenis susunan warna window. Struktur data dari WindowStyle ini dapat dilihat pada file SHAPE.H. Berikut ini adalah cuplikan sebagian dari file SHAPE.H.

```
#define MAXWINDOW 50

typedef struct
{
  char R,G,B;
} RGBCol;
```

Pada masing-masing RGBCol, disimpan tiga buah unsur warna.

```
typedef struct
{
  RGBCol inframe;
  RGBCol midframe;
  RGBCol outframe;
  RGBCol bgr;
  RGBCol bound;
```

Pada _WindowStyle disimpan 5 set warna untuk menyimpan warna bingkai terluar, tengah dan dalam dari window, pembatas antara background dengan bingkai dan juga warna background.

```
} _WindowStyle;

_WindowStyle WindowStyle[MAXWINDOW];
```

Deklarasikan _WindowStyle sebagai array sebesar 50 cell sehingga kita dapat menyimpan 50 macam model warna window.

Tujuan digunakannya struktur data ini ialah agar kita tidak perlu selalu mengirimkan parameter yang berisi setiap unsur warna dari setiap bingkai di sebuah window pada saat menampilkan window tersebut. Pendefinisian warna cukup dilakukan sekali dengan fungsi SetWindowStyle kemudian kita cukup menggambar window dengan memberitahukan style (nomor array) mana yang akan kita gunakan sebagai tata warna dari window.

Fungsi SetWindowStyle

Fungsi ini digunakan untuk mendefinisikan tata warna dari sebuah cell window.

```
void SetWindowStyle(long style,RGBCol inframe,RGBCol mid-
frame,RGBCol outframe,RGBCol bgr,RGBCol bound)
{
```

Parameter yang dikirimkan ialah style yaitu nomor cell array yang akan digunakan sebagai penyimpan tata warna window. Kemudian inframe, midframe, outframe, bgr dan bound yang menyimpan tata warna dari bingkai window.

```
  WindowStyle[style].inframe=inframe;
  WindowStyle[style].midframe=midframe;
  WindowStyle[style].outframe=outframe;
  WindowStyle[style].bgr=bgr;
  WindowStyle[style].bound=bound;
```

Isikan tata warna ke dalam array WindowStyle pada index style.

```
}
```

Cobalah program TUNIT.EXE yang akan mencoba fungsi penggambaran window pada virtual screen.

Pada saat Anda jalankan, program ini akan menggambar background kotak-kotak kecil sama seperti pada program TBLOCK.EXE, kemudian akan ditampilkan sebuah window pada sisi kanan layar. Selanjutnya akan terlihat sebuah titik bergerak dari kiri atas layar ke kanan atas layar. Kecepatan pergerakan titik ini menentukan kecepatan frame per second dari sistem komputer Anda dalam mode graphic 640 x 480 24 bit dengan menggunakan unit 24BIT.ASM.

PENJELASAN PROGRAM

```
/*-------------------------------------------------------
 Nama        : TUNIT.CPP
 Definisi    : Program mencoba fungsi window
 Programmer : Agustinus Nalwan
------------------------------------------------------- */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
```

Definisikan unit SHAPE.CPP beserta file headernya.

```
char * buffer;

void test()
{
  long x,y,ctr;
  long r=0,g=0,b=0;
  RGBCol inf,outf,midf,bgr,bound;
```

Definisikan inf, outf, midf, bgr dan bound untuk tata warna window.

```
  for (x=0;x<128;x++)
  {
    for (y=0;y<96;y++)
    {
      Block(buffer,x*5,y*5,x*5+4,y*5+4,r,g,b);
      b+=2;
```

```
    g+=2;
  }
  b=0;
  g=0;
  r+=2;
}
```

Tampilkan background kotak-kotak sama seperti pada program TBLOCK.EXE.

```
x=0;y=0;
inf.R=156; inf.G=179;   inf.B=196;
midf.R=126; midf.G=149; midf.B=166;
outf.R=96; outf.G=119; outf.B=136;
bound.R=66; bound.G=89; bound.B=106;
bgr.R=96; bgr.G=119;   bgr.B=136;
```

Isi variabel-variabel inf, midf, outf, bound dan bgr dengan intensitas R, G, B yang kita inginkan.

```
SetWindowStyle(0,inf,midf,outf,bgr,bound);
```

Kirimkan variabel-variabel tersebut sebagai tata warna window nomor 0.

```
DrawWindow(buffer,550,0,639,479,0);
while (!kbhit())
{
  PutPixel(buffer,x,y,0xF0F0F0);
  DrawWindow(buffer,550,0,639,479,0);
```

Tampilkan titik pada virtual screen dengan koordinat x,y dan susunan warna gabungan $F0F0F0 yang berarti R = 128, G = 128 dan B = 128.

Tampilkan window pada koordinat (550,0)-(639,479) dengan nomor style = 0 yang telah kita rakit sebelumnya.

```
  x+=1;
  if (x>639)
  {
    y+=1;
    x=0;
  }
```

Gerakkan titik dari kiri atas ke kanan bawah layar.

```
  MoveToScreen(buffer);
```

Tampilkan virtual screen ke layar monitor.

```
  }
}

int main()
{
  OpenVGA24BIT(MOD640);
```

Buka mode graphic 640 x 480 24 bit.

```
  MaxX=639;MaxY=479;
  buffer=(char *)malloc(MEM640);
```

Deklarasikan virtual memory untuk mode graphic tersebut.

```
  test();
```

Lakukan fungsi penggambaran di atas.

```
  CloseVGA24BIT();
```

Tutup mode graphic.

```
  return 1;
}
```

7.3 Penampilan Gambar 8 Bit ke Mode 24 Bit

Bagaimana cara mengkonversi gambar yang sudah telanjur dibuat dalam mode graphic 8 bit ke mode graphic 24 bit? Apakah kita harus menggambar ulang dan menyimpannya dalam format 24 bit? Tidak perlu!

Ciri khas gambar dengan format 8 bit ialah digunakannya tabel palette dan tiap titik merupakan indeks dari tabel palette. Pada format 24 bit, setiap titik memiliki intensitas R, G dan B sendiri sehingga untuk mengubah dari format 8 bit ke 24 bit, kita cukup membaca susunan warna yang dipakai oleh setiap titik pada mode 8 bit kemudian menuliskan susunan warna ini secara langsung sebagai 3 byte pada setiap titiknya. Coba jalankan program TPIC.EXE. Program ini akan menampilkan file gambar dengan format 8 bit ke virtual screen dengan mode graphic 24 bit.

PENJELASAN PROGRAM

```
/*-------------------------------------------------------
 Nama        : TPIC.CPP
 Definisi    : Program untuk mencoba menampilkan gambar
               256 warna ke 24 bit
 Programmer : Agustinus Nalwan
------------------------------------------------------- */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
```

Definisikan library SHAPE.CPP.

Definisikan library VGAPAL.CPP. Library ini sama persis dengan library VGAPAL yang kita gunakan pada pembahasan graphic 8 bit. Untuk apa? Library ini digunakan untuk membaca susunan palette yang dipergunakan oleh gambar format 8 bit yang akan ditampilkan pada mode 24 bit.

```
char * buffer;
char ch;

void LoadPicture(char * fname, char * Buff)
{
  FILE * fpic;
  int x,y;
  char col;
  int r,g,b;
  fpic=fopen(fname,"rb");
```

Baca file gambar.

```
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
```

Lakukan loop untuk mengakses setiap titik.

```
      fread(&col,1,1,fpic);
```

Baca nomor warna dari titik.

```
r=*(devPal+col*3);
g=*(devPal+col*3+1);
b=*(devPal+col*3+2);
```

Gunakan nomor warna titik sebagai indeks dari tabel palette untuk membaca susunan intensitas R, G dan B yang sebenarnya dari titik yang sedang diakses.

Susunan warna ini didapat dari alamat memory devPal di mana file palette telah diisikan pada pemanggilan Install_Palette.

```
_PutPixel(Buff,x,y,r*4,g*4,b*4);
```

Tampilkan titik tersebut pada virtual screen sesuai dengan harga R, G dan B-nya yang dikalikan 4. Kenapa? karena intensitas maksimum dari setiap R, G dan B pada mode 8 bit ialah 31 (32 intensitas keterangan). Pada mode 24 bit, setiap R, G dan B memiliki intensitas keterangan 256 yaitu 4 kali lipat dari mode 8 bit sehingga untuk memperoleh hasil yang sama, susunan warna harus dikalikan 4.

```
    }
  }
  fclose(fpic);
```

Tutup file gambar.

```
}

int main()
{
  OpenVGA24BIT(MOD640);
```

Buka mode graphic 640 x 480 24 bit.

```
  MaxX=639;MaxY=479;
  buffer=(char *)malloc(MEM640);
  ClearDouble(buffer,0);
```

Alokasikan memory untuk virtual screen.

```
  Install_Palette("24bspr.pal");
```

Baca file palette yang digunakan oleh gambar 24BSPR.PIC.

```
  LoadPicture("24bspr.pic",buffer);
  MoveToScreen(buffer);
```

Baca file gambar 24BSPR.PIC dan tampilkan dalam mode 24 bit pada virtual screen buffer.

```
  ch=getch();
  CloseVGA24BIT();
```

Tutup mode graphic.

```
  return 1;
}
```

Dengan mode graphic 24 bit, variasi warna yang dapat kita ciptakan jauh lebih banyak dibanding dengan mode 8 bit. Kita dapat mengatur warna setiap titik semau kita tanpa perlu takut berubahnya warna pada titik-titik yang lain. Nah, sekarang cobalah program TCOLOR.EXE yang merupakan modifikasi dari program di atas dengan memainkan efek warna sehingga diperoleh variasi warna yang bermacam-macam.

```
/*-------------------------------------------------------
 Nama        : TCOLOR.CPP
 Definisi    : Program untuk mencoba penampilan gambar
               8 bit dengan efek warna dalam 24 bit
 Programmer : Agustinus Nalwan
------------------------------------------------------- */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
```

Definisikan penggunaan unit VGAPAL.CPP.

```
char * buffer;
char ch;

void LoadPicture(int sx,int sy, float ir,float ig,float ib,char *
fname, char * Buff)
{
```

Sekarang parameter yang dikirimkan untuk fungsi LoadPicture bertambah.

(sx,sy) ialah koordinat di mana gambar akan diletakkan. Karena

gambar pada mode 8 bit yang akan dibaca hanya berukuran 320 x 200 sedangkan layar kita berukuran 640 x 480.

Ir, Ig dan Ib ialah tiga konstanta yang nantinya akan digunakan untuk memainkan intensitas R, G dan B dari gambar.

Fname ialah nama file gambar yang akan dibaca dan Buff ialah virtual screen tempat gambar akan ditampilkan.

```
FILE * fpic;
int x,y;
char col;
int r,g,b;
fpic=fopen(fname,"rb");
```

Buka file gambar.

```
for (y=0;y<200;y++)
{
  for (x=0;x<320;x++)
  {
```

Lakukan loop untuk memproses seluruh titik.

```
    fread(&col,1,1,fpic);
```

Baca nomor warna dari titik.

```
    r=*(devPal+col*3);
    g=*(devPal+col*3+1);
    b=*(devPal+col*3+2);
```

Ambil nilai R, G dan B sebenarnya dari titik pada tabel palette.

```
    _PutPixel(Buff,x+sx,y+sy,r*ir,g*ig,b*ib);
```

Tampilkan titik tersebut mulai pada koordinat kiri atas (sx,sy) dan gunakan konstanta ir, ig dan ib untuk menentukan proporsi dari setiap intensitas warna. Dengan ini, kita dapat memainkan intensitas mana yang dominan. Jika kita ingin gambar kita terlihat lebih merah, kita cukup memberikan harga yang besar pada ir dan harga yang kecil pada ig dan ib.

```
  }
}
fclose(fpic);
```

Tutup file gambar.

```
}

int main()
{
  OpenVGA24BIT(MOD640);
  MaxX=639;MaxY=479;
  buffer=(char *)malloc(MEM640);
  ClearDouble(buffer,0);
  Install_Palette("24bspr.pal");
  LoadPicture(0,0,1,1,4,"24bspr.pic",buffer);
  LoadPicture(320,1,4,4,4,"24bspr.pic",buffer);
  LoadPicture(0,200,1,4,1,"24bspr.pic",buffer);
  LoadPicture(320,200,4,1,4,"24bspr.pic",buffer);
```

Sekarang gambar ditampilkan empat kali dengan koordinat yang berlainan dan dengan proporsi R, G dan B yang berlainan pula sehingga menghasilkan efek yang lain pada setiap penampilannya. Pada peletakan pertama terlihat warna biru lebih dominan (Ir = 1, Ig = 1 dan Ib = 4). Penampilan kedua terlihat seluruh warna dominan (Ir = 4, Ig = 4 dan Ib = 4).

```
  MoveToScreen(buffer);
```

Tampilkan virtual screen ke layar monitor.

```
  ch=getch();
  CloseVGA24BIT();
```

Tutup mode graphic 24 bit.

```
  return 1;
}
```

Penampilan sprite dan efek-efek warna pada mode 24 bit.

Berikutnya akan kita bahas mengenai penampilan sprite pada mode 24 bit. Fungsi-fungsi yang akan kita bahas di sini tidak sebanyak fungsi penampilan sprite pada mode graphic 8 bit terdahulu. Tetapi fungsi yang ada pada library 24 bit ini dapat menghasilkan efek yang tidak mungkin kita ciptakan dengan mode graphic 8 bit. Fungsi tersebut ialah fungsi Put24BitMask yang memiliki variasi penampilan sprite yang dapat kita atur. Ingin tahu lebih lanjut mengenai fungsi ini? Teruskan membaca....

7.4 Fungsi Penampilan Sprite Standar

Fungsi standar untuk penampilan sprite ialah fungsi untuk merekam sprite yaitu GetImage dan fungsi untuk meletakkan sprite pada virtual screen.

Fungsi-fungsi ini beserta fungsi-fungsi spesial lainnya dirancang dalam file IMAGE.CPP.

Fungsi GetImage

Fungsi ini digunakan untuk merekam gambar dalam mode 24 bit pada virtual screen ke dalam alamat memory kosong. Format penyimpanan sprite ini mirip dengan fungsi GetImage pada 8 bit, hanya saja di sini setiap titik menyimpan tiga byte yang masing-masing berisi intensitas R, G dan B.

Sehingga untuk menampung gambar dengan ukuran yang sama, mode 24 bit membutuhkan tempat penyimpanan 3 kali lipat dari mode 8 bit.

```
void GetImage(char * Buff,int x1,int y1,int x2,int y2,char * Sprite)
{
```

Buff ialah virtual screen di mana sprite berada. (x1,y1)-(x2,y2) adalah koordinat kiri atas dan kanan bawah dari sprite yang akan direkam. Sprite akan disimpan dalam alamat memory Sprite.

```
  int a,b;
  int ctr;
  int br,bg,bb;
  char temp[4];
  temp[0]=(x2-x1+1)%256;
  temp[1]=(x2-x1+1)/256;
  temp[2]=(y2-y1+1)%256;
  temp[3]=(y2-y1+1)/256;
  memcpy(Sprite,temp,4);
```

Isikan 2 byte pertama sebagai panjang sprite. Dan 2 byte berikutnya sebagai lebar sprite.

```
  ctr=4;
  for (b=y1;b<=y2;b++)
  {
    for (a=x1;a<=x2;a++)
    {
```

Lakukan loop untuk merekam seluruh titik.

```
bb=*(Buff+b*(MaxX+1)*3+a*3);
bg=*(Buff+b*(MaxX+1)*3+a*3+1);
br=*(Buff+b*(MaxX+1)*3+a*3+2);
```

Ambil intensitas R, G dan B dari titik yang sedang diproses pada virtual screen. Untuk pengambilan intensitas B, koordinat y yaitu variabel b dikalikan dengan (MaxX+1)*3 yaitu jumlah byte horizontal, kemudian ditambah dengan koordinat x yaitu a yang dikalikan dengan 3. Perkalian dengan tiga ini karena setiap titik disimpan dalam 3 byte. Untk intensitas G dan B adalah dua byte berikutnya.

```
*(Sprite+ctr)=br;
*(Sprite+ctr+1)=bg;
*(Sprite+ctr+2)=bb;
```

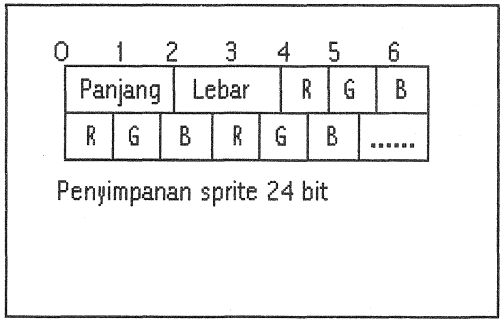
Letakkan ketiga intensitas tersebut dalam alamat memory Sprite.

```
ctr+=3;
```

Tambahkan variabel counter untuk memproses titik berikutnya.

```
    }
  }
}
```

Format sprite 24 bit ini dapat kita lihat pada Gambar 7.2.



Gambar 7.2 Model Penyimpanan Sprite 24 Bit

Sehingga untuk mencari offste dari sebuah titik dengan indeks horizontal x dan indeks vertikal y ialah:

offset = Y * Panjang* 3 + X * 3 + 4

Fungsi GetImageDimension

Fungsi ini digunakan untuk mengambil panjang dan lebar sebuah sprite.

```
void GetImageDimension(char * Pic, int & x,int & y)
{
```

Parameter yang dikirimkan ialah Pic yaitu alamat memory tempat sprite disimpan. X dan Y ialah variabel output yang akan memberikan informasi panjang dan lebar sprite tersebut.

```
  char temp[4];
  int a,b,c,d;
  memcpy(temp,Pic,4);
```

Kopikan informasi panjang dan lebar (4 byte pertama) dari sprite ke array.

```
  b=temp[0];
  a=temp[1];
  d=temp[2];
  c=temp[3];
  x=a*256+b;
  y=c*256+d;
```

Gabung dua byte pertama array menjadi panjang dari sprite dan dua byte berikutnya menjadi lebar dari sprite.

```
}
```

Fungsi Put24Bit

Fungsi ini ialah fungsi yang digunakan untuk menampilkan sprite pada virtual screen. Format sprite yang digunakan di sini ialah 24 bit. Nah, sekarang yang menjadi masalah, warna apa yang akan kita gunakan sebagai warna yang tidak ditampilkan? Pada mode 8 bit, kita anggap warna 0 sebagai warna yang tidak perlu ditampilkan ke layar. Di sini kita dapat menganggap titik yang warnanya memiliki intensitas R = 0, G = 0 dan B = 0 adalah titik yang tidak perlu ditampilkan. Kelemahan teknik ini ialah kita tidak dapat mempergunakan warna R = 0, G = 0 dan B = 0 sebagai warna titik dari sprite, karena warna ini tidak akan ditampilkan. Oleh karena itu, kita menggunakan warna yang paling dekat dengan warna ini yaitu R = 1, G = 1 dan B = 1, toh karena bedanya terlalu kecil hampir tidak terlihat oleh mata manusia.

```
void Put24Bit(char * Buff,long x,long y,char * Pic)
{
```

Parameter yang dikirimkan ialah Buff yaitu alamat memory virtual screen di mana sprite akan diletakkan. (x,y) ialah koordinat di mana virtual screen akan diletakkan dan Pic ialah alamat memory di mana sprite disimpan.

```
  int a,b;
  int Rr,Gg,Bb;
  int x1,y1,x2,y2,skipx,skipy;
  long rx,ry,length,cx;
  char cread;
  skipx=0;skipy=0;
  x1=x;y1=y;
  GetImageDimension(Pic,x2,y2);
```

Ambil panjang dan lebar dari sprite dengan menggunakan fungsi GetImageDimension.

```
  length=x2;
  x2+=x1;y2+=y1;
  x2-=1;y2-=1;
```

Hitung koordinat kanan bawah dari sprite.

```
  if (x1<0)
  {
    x1=0;
    skipx=-x;
  }
  if (y1<0)
  {
    y1=0;
    skipy=-y;
  }
```

Potong sprite terhadap kiri atas layar.

```
  if (x2>MaxX)
  {
    x2=MaxX;
  }
  if (y2>MaxY)
  {
    y2=MaxY;
  }
```

Potong sprite terhadap kanan bawah layar.

```
  if (x1<=x2&&y1<=y2&&x1<=MaxX&&y1<=MaxY)
  {
```

Jika sprite masih berada dalam batas layar, teruskan penampilan sprite.

```
    for (a=x1;a<=x2;a++)
    {
      for (b=y1;b<=y2;b++)
      {
```

Lakukan loop untuk seluruh titik pada sprite.

```
        rx=a-x;ry=b-y;
```

Hitung indeks horizontal dan indeks vertikal dari titik sprite yang sedang diproses. Indeks ini nantinya akan digunakan untuk menghitung offset dari titik yang sedang diproses.

```
        cx=ry*length*3+rx*3;
```

Hitung offset dari titik yang sedang diproses dengan rumus perhitungan offset mode 24 bit.

```
        Rr=*(Pic+cx+4);
        Gg=*(Pic+cx+5);
        Bb=*(Pic+cx+6);
        if (Rr==0&&Gg==0&&Bb==0); else
          _PutPixel(Buff,a,b,Rr,Gg,Bb);
```

Jika titik memiliki susunan warna R = 0 dan G = 0 dan B = 0 maka titik tersebut tidak perlu ditampilkan.

```
      }
    }
  }
}
```

Untuk mencoba fungsi di atas, coba jalankan program TSPRITE.EXE.

PENJELASAN PROGRAM

```
/*-------------------------------------------------------
 Nama        : TSPRITE.CPP
 Definisi    : Program untuk mencoba penampilan sprite
 Programmer  : Agustinus Nalwan
------------------------------------------------------- */
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
#include "image.h"
#include "image.cpp"
```

Definisikan unit SHAPE.CPP.

Definisikan unit VGAPAL.CPP.

Definisikan unit IMAGE.CPP.

```
char * buffer;
char * sprite;
char ch;

void LoadPicture(char * fname, char * Buff)
{
  FILE * fpic;
  int x,y;
  char col;
  int r,g,b;
  fpic=fopen(fname,"rb");
```

Buka file gambar.

```
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
```

Lakukan loop untuk memproses seluruh titik pada file gambar.

```
      fread(&col,1,1,fpic);
```

Ambil warna titik.

```
      r=*(devPal+col*3);
      g=*(devPal+col*3+1);
      b=*(devPal+col*3+2);
```

Baca intensitas R, G, B sebenarnya dari titik.

```
      _PutPixel(Buff,x,y,r*4,g*4,b*4);
```

Tampilkan ke layar.

```
    }
  }
  fclose(fpic);
}

int main()
{
  OpenVGA24BIT(MOD640);
  MaxX=639;MaxY=479;
  buffer=(char *)malloc(MEM640);
  sprite=(char *)malloc(320*200*3+4);
```

Alokasikan memory untuk virtual screen buffer dan untuk sprite. Penggunaan memory untuk sprite dihitung dengan rumus panjang x lebar x 3 + 4. Di mana perkalian dengan 3 ialah karena setiap titik menggunakan 3 byte memory, dan penambahan dengan 4 dilakukan karena 4 byte pertama digunakan untuk menyimpan informasi panjang dan lebar dari sprite.

```
  ClearDouble(buffer,0);
  Install_Palette("24bspr.pal");
  LoadPicture("24bspr.pic",buffer);
```

Instal palette dan baca gambar.

```
  GetImage(buffer,0,0,319,199,sprite);
```

Rekam gambar dari koordinat (0,0)-(319,199) ke alamat memory sprite.

```
  Put24Bit(buffer,320,0,sprite);
```

Tampilkan sprite ke virtual screen buffer pada koordinat (320,0).

```
  MoveToScreen(buffer);
```

Tampilkan virtual screen ke layar monitor.

```
  ch=getch();
  CloseVGA24BIT();
```

Tutup mode graphic 24 bit.

```
  return 1;
}
```

7.5 Efek Gamma pada Sprite

Fungsi selanjutnya yang akan kita bahas ialah Put24BitMask. Fungsi ini dirancang untuk menghasilkan tiga macam efek peletakan gambar. Yang pertama ialah peletakan sprite dengan efek gamma yaitu kita dapat mengubah dominasi dari setiap intensitas warna (R, G, B). Hal ini sama persis dengan apa yang kita lakukan pada program TCOLOR.EXE hanya saja efek penggantian warna tersebut sekarang kita terapkan pada saat penampilan gambar.

Coba jalankan program TGAMMA.EXE.

Akan ditampilkan 4 buah muka patung dengan dominasi warna yang berbeda pada setiap gambarnya. Gambar asli yang disimpan dalam sprite ialah gambar pertama yaitu gambar kiri atas. Gambar-gambar yang lain ialah hasil permainan parameter dari fungsi Put24BitMask dengan efek gamma.

Yuk, kita lihat listing program dari TGAMMA.CPP.

```
/*-------------------------------------------------------
 Nama        : TGAMMA.CPP
 Definisi    : Program untuk mencoba efek gamma pada sprite
 Programmer : Agustinus Nalwan
------------------------------------------------------- */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
#include "image.h"
#include "image.cpp"
```

Definisikan unit SHAPE.CPP.

Definisikan unit VGAPAL.CPP.

Definisikan unit IMAGE.CPP.

```
char * buffer;
char * sprite;
char ch;

void LoadPicture(char * fname, char * Buff)
{
```

```
  FILE * fpic;
  int x,y;
  char col;
  int r,g,b;
  fpic=fopen(fname,"rb");
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
      fread(&col,1,1,fpic);
      r=*(devPal+col*3);
      g=*(devPal+col*3+1);
      b=*(devPal+col*3+2);
      _PutPixel(Buff,x,y,r*4,g*4,b*4);
    }
  }
  fclose(fpic);
```

Baca file gambar ke virtual screen.

```
}

int main()
{
  OpenVGA24BIT(MOD640);
  MaxX=639;MaxY=479;
  buffer=(char *)malloc(MEM640);
  sprite=(char *)malloc(320*200*3+4);
```

Alokasikan memory untuk virtual screen dan untuk sprite.

```
  ClearDouble(buffer,0);
  Install_Palette("24bspr.pal");
  LoadPicture("24bspr.pic",buffer);
```

Baca file palette dan file gambar.

```
  GetImage(buffer,0,0,319,199,sprite);
```

Rekam potongan gambar dalam sprite.

```
  Put24BitMask(buffer,320,0,sprite,1,50,50,1);
```

Tampilkan sprite ke virtual screen dengan koordinat (320,0) dan dengan nomor efek = 1 (efek gamma) dan dengan proporsi intensitas Ir = 1/50, Ig = 1/50 dan Ib = 1/1.

Proporsi Ir, Ig dan Ib ini nantinya akan dikalikan dengan R, G dan B dari titik yang akan diletakkan. Sehingga semakin besar angka yang

kita kirimkan sebagai parameter di atas, semakin hilang dominasi dari intensitas warna tersebut.

```
Put24BitMask(buffer,0,200,sprite,1,1,50,50);
Put24BitMask(buffer,320,200,sprite,1,50,1,50);
```

Tampilkan sprite dua kali lagi dengan koordinat dan proporsi dominasi intensitas yang berbeda.

```
MoveToScreen(buffer);
```

Tampilkan virtual screen ke layar monitor.

```
ch=getch();
CloseVGA24BIT();
```

Tutup mode graphic 24 bit.

```
  return 1;
}
```

Nah, sekarang coba kita lihat fungsi Put24BitMask yang dipergunakan untuk menghasilkan efek-efek tersebut.

```
void      Put24BitMask(char * Buff,long x,long y,char *
Pic,int model,float ir,float ig,float ib)
{
```

Parameter yang dikirimkan ialah:

```
Buff      = Alamat virtual screen yang akan digambari.
X,Y       = Ialah koordinat sprite pada virtual
            screen.
Pic       = Ialah sprite yang akan ditampilkan.
Model     = Ialah model efek yang dipilih.
              1 = Efek gamma.
              2 = Efek transparansi.
              3 = Efek cahaya.
Ir,Ig,Ib  = Proporsi dominasi dari setiap intensitas.
            Kegunaan dari ketiga variabel ini berbeda
            untuk jenis efek yang berbeda.
```

```
  int a,b;
  int x1,y1,x2,y2,skipx,skipy;
  float resr,resg,resb;
  float bgr,bgg,bgb;
  float rr,gg,bb;
```

```
long rx,ry,length,cx;
char cread;
skipx=0;skipy=0;
x1=x;y1=y;
GetImageDimension(Pic,x2,y2);
length=x2;
x2+=x1;y2+=y1;
x2-=1;y2-=1;
```

Hitung koordinat kanan bawah dari sprite.

```
if (x1<0)
{
  x1=0;
  skipx=-x;
}
if (y1<0)
{
  y1=0;
  skipy=-y;
}
```

Potong sprite terhadap kiri atas layar.

```
if (x2>MaxX)
{
  x2=MaxX;
}
if (y2>MaxY)
{
  y2=MaxY;
}
```

Potong sprite terhadap kanan bawah layar.

```
if (x1<=x2&&y1<=y2&&x1<=MaxX&&y1<=MaxY)
{
```

Jika sprite masih berada dalam layar, teruskan proses penggambaran sprite.

```
  if (model==1)
  {
```

Jika model = 1 (efek gamma).

```
    for (a=x1;a<=x2;a++)
    {
```

```
for (b=y1;b<=y2;b++)
{
```

Lakukan loop untuk memproses seluruh titik dalam sprite.

```
rx=a-x;ry=b-y;
cx=ry*length*3+rx*3;
```

Hitung offset dari titik yang sedang diakses.

```
rr=*(Pic+cx+4);
gg=*(Pic+cx+5);
bb=*(Pic+cx+6);
```

Ambil intensitas R, G dan B dari titik yang sedang diakses.

```
if (rr==0&&gg==0&&bb==0); else
```

Jika titik harus ditampilkan, teruskan proses di bawah ini.

```
_PutPixel(Buff,a,b,rr/ir,gg/ig,bb/ib);
```

Bagi intensitas R, G dan B dengan konstanta ir, ig dan ib sehingga semakin besar harga ir maka semakin kecil dominasi intensitas R (merah).

```
        }
    }
}
```

Baris-baris program berikutnya dari fungsi Put24BitMask ini akan kita bahas setelah kita mencoba efek transparansi.

7.6 Efek Multi Channel Transparansi pada Sprite

Efek ini sebenarnya dibuat dengan dasar teknik transparansi yang telah kita pelajari sebelumnya pada pemrograman graphic library 8 bit. Yang dimaksud dengan Multi Channel di sini ialah kita dapat mengatur proporsi dominasi dari setiap intensitas warna R, G dan B dalam proses transparansi. Sehingga kita dapat membuat hasil transparansi agak berwarna kemerah-merahan/kebiru-biruan dengan memainkan konstanta proporsi dominasi ini.

Coba jalankan program TTRANS.EXE. WOW keren khan? Coba tekan tombol:

- q/a untuk menambah/mengurangi proporsi warna merah.
- W/s untuk menambah/mengurangi proporsi warna hijau.
- E/d untuk menambah/mengurangi proporsi warna biru.
- Kursor atas/bawah untuk menambah/mengurangi ketiga unsur warna sekaligus.
- Tekan ESC untuk keluar.

Jika Anda tekan kursor bawah terus menerus akan menyebabkan gambar semakin lama semakin samar dan lenyap. Hal ini karena proporsi dari ketiga unsur warna sangat kecil dan lama-lama berharga 0 sehingga tidak menimbulkan efek pada background.

Coba bayangkan jika kita harus menciptakan efek ini pada mode 8 bit. Berapa jumlah tabel yang harus kita buat. Pokoknya hampir tidak mungkin.

```
/*-------------------------------------------------------
 Nama        : TTRANS.CPP
 Definisi    : Program untuk mencoba efek transparansi
 Programmer : Agustinus Nalwan
------------------------------------------------------- */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
#include "image.h"
#include "image.cpp"

char * buffer;
char * back;
char * sprite;
char ch;

void LoadPicture(int mode,char * fname, char * Buff)
{
```

Fungsi LoadPicture ini kita ganti sedikit dengan menggunakan parameter mode. Parameter ini jika berharga 1 maka ukuran gambar yang akan ditampilkan akan diperbesar 4 kali lipat sehingga dapat memenuhi layar 640 x 480 titik. Jika parameter mode berharga bukan 1 maka ukuran gambar akan ditampilkan normal yaitu 320 x 200 titik.

```
FILE * fpic;
int x,y;
char col;
int r,g,b;
fpic=fopen(fname,"rb");
for (y=0;y<200;y++)
{
  for (x=0;x<320;x++)
  {
```

Lakukan loop untuk membaca seluruh titik.

```
    fread(&col,1,1,fpic);
```

Baca nomor warna titik.

```
    r=*(devPal+col*3);
    g=*(devPal+col*3+1);
    b=*(devPal+col*3+2);
```

Ambil intensitas R, G dan B dari titik.

```
    if (mode==1)
    {
      _PutPixel(Buff,x*2,y*2,r*4,g*4,b*4);
      _PutPixel(Buff,x*2+1,y*2,r*4,g*4,b*4);
      _PutPixel(Buff,x*2,y*2+1,r*4,g*4,b*4);
      _PutPixel(Buff,x*2+1,y*2+1,r*4,g*4,b*4);
```

Pada mode 1 tampilkan titik 4 kali lebih besar.

```
    } else
      _PutPixel(Buff,x,y,r*4,g*4,b*4);
```

Pada mode lain tampilkan titik dengan ukuran biasa.

```
    }
  }
  fclose(fpic);
}
```

```
int main()
{
  float ir,ig,ib;
  int finish=0;
  ir=1;ig=1;ib=1;
  OpenVGA24BIT(MOD640);
  MaxX=639;MaxY=479;
  buffer=(char *)malloc(MEM640);
  back=(char *)malloc(MEM640);
  sprite=(char *)malloc(200000);
```

Alokasikan memory untuk virtual screen buffer dan back dan sprite.

```
  ClearDouble(buffer,0);
  Install_Palette("24bspr.pal");
  LoadPicture(0,"24bspr.pic",buffer);
```

Baca file gambar dan palette. Gunakan mode 0 sehingga gambar ditampilkan seukuran normal. Gambar ini menampung gambar yang akan kita jadikan sprite.

```
  GetImage(buffer,0,0,319,199,sprite);
  free(devPal);
  free(actPal);
```

Rekam sprite dan hapus tabel palette karena kita akan membaca susunan palette yang baru.

```
  Install_Palette("town.pal");
  ClearDouble(back,0);
  LoadPicture(1,"town.pic",back);
```

Baca file gambar dan masukkan ke virtual screen back sebagai background. Susunan palette gambar ini berada pada file TOWN.PAL.

```
  while(!finish)
  {
    MoveDouble(back,buffer);
    Put24BitMask(buffer,160,100,sprite,2,ir,ig,ib);
```

Tampilkan sprite pada koordinat (160,100) dengan efek nomor 2 (transparansi) dengan proporsi dominasi ir, ig dan ib.

```
    MoveToScreen(buffer);
```

Tampilkan virtual screen ke layar monitor.

```
    while (!kbhit())
    {
    }
    ch=getch();
    if (ch==27) finish=1;
    if (ch=='q') ir+=0.2;
    if (ch=='a') ir-=0.2;
    if (ch=='w') ig+=0.2;
    if (ch=='s') ig-=0.2;
    if (ch=='e') ib+=0.2;
    if (ch=='d') ib-=0.2;
```

Ubah harga ir, ig dan ib sesuai dengan penekanan tombol keyboard.

```
    if (ch==72)
    {
      ir+=0.2;ig+=0.2;ib+=0.2;
    }
    if (ch==80)
    {
      ir-=0.2;ig-=0.2;ib-=0.2;
    }
```

Ganti harga ketiga unsur ini sekaligus pada penekanan tombol kursor atas/bawah.

```
    if (ir<0) ir=0;
    if (ir>10) ir=10;
    if (ig<0) ig=0;
    if (ig>10) ig=10;
    if (ib<0) ib=0;
    if (ib>10) ib=10;
```

Batasi konstanta tersebut sehingga tidak melebihi 10 dan kurang dari 0.

```
  }
  CloseVGA24BIT();

Tutup mode graphic 24 bit.
  return 1;
}
```

Bagian dari fungsi Put24BitMask yang menangani efek ini akan kita bahas di bawah ini.

```
    if (model==2)
    {
```

Jika model = 2 maka efek yang dipilih ialah efek transparansi.

```
for (a=x1;a<=x2;a++)
{
  for (b=y1;b<=y2;b++)
  {
```

Lakukan loop untuk mengakses seluruh titik pada sprite.

```
rx=a-x;ry=b-y;
cx=ry*length*3+rx*3;
```

Hitung offset dari titik yang sedang diakses.

```
rr=*(Pic+cx+4);
gg=*(Pic+cx+5);
bb=*(Pic+cx+6);
```

Baca intensitas R, G dan B dari titik yang sedang diakses.

```
bgb=*(Buff+b*(MaxX+1)*3+a*3);
bgg=*(Buff+b*(MaxX+1)*3+a*3+1);
bgr=*(Buff+b*(MaxX+1)*3+a*3+2);
```

Baca intensitas R, G dan B dari titik di bawah titik yang sedang diakses pada virtual screen.

```
resr=(rr*ir+bgr);resg=(gg*ig+bgg);
resb=(bb*ib+bgb);
```

Kalikan intensitas titik sprite dengan konstanta ir dan tambahkan dengan intensitas titik pada virtual screen di bawah titik sprite yang akan diletakkan. Dengan ini diperoleh penggabungan antara dua warna titik sehingga diperoleh efek transparansi. Jika konstanta ir semakin besar maka dominasi rr akan semakin besar sehingga unsur merah dari sprite akan semakin mendominasi hasil transparansi. Demikian juga pada unsur hijau dan biru.

```
if (resr>255) resr=255;
if (resg>255) resg=255;
if (resb>255) resb=255;
```

Batasi hasil transparansi pada 255 sebagai intensitas maksimum.

```
if (rr==0&&gg==0&&bb==0); else
  _PutPixel(Buff,a,b,resr,resg,resb);
```

Jika titik harus ditampilkan maka tampilkan titik dengan fungsi _PutPixel.

```
        }
      }
    }
```

7.7 Efek Sprite Cahaya

Kalau pada mode 8 bit kita memiliki fungsi VESA_Lightning, pada mode ini kita juga tidak mau kalah. Bahkan efek cahaya pada mode 24 bit jauh lebih bagus dibandingkan dengan mode 8 bit, karena jumlah warna yang jauh lebih banyak dan dapat diaksesnya warna per titik tanpa perlu mengubah intensitas warna titik yang lain.

Coba jalankan program TLIGHT.EXE.

Bagaimana? Puas? Pemrograman graphic di 24 bit memang benar-benar menyenangkan karena hasil yang kita dapat tidak tanggung-tanggung. Mode graphic 24 bit ini kadang bahkan memberikan hasil yang jauh lebih bagus dari tampilan TV jika digunakan resolusi tinggi.

Di sini Anda juga dapat menekan tombol-tombol yang sama seperti pada program TTRANS.EXE untuk mengubah proporsi dominasi dari setiap unsur sehingga didapat intensitas cahaya yang berubah warna.

PENJELASAN PROGRAM

```
/*-------------------------------------------------------
 Nama        : TLIGHT.CPP
 Definisi    : Program untuk mencoba efek sprite cahaya
 Programmer : Agustinus Nalwan
------------------------------------------------------- */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "24bit.h"
#include "shape.h"
#include "shape.cpp"
#include "vgapal.h"
#include "vgapal.cpp"
#include "image.h"
#include "image.cpp"

char * buffer;
```

```
char * back;
char * sprite;
char ch;

void LoadPicture(int mode,char * fname, char * Buff)
{
  FILE * fpic;
  int x,y;
  char col;
  int r,g,b;
  fpic=fopen(fname,"rb");
  for (y=0;y<200;y++)
  {
    for (x=0;x<320;x++)
    {
      fread(&col,1,1,fpic);
      r=*(devPal+col*3);
      g=*(devPal+col*3+1);
      b=*(devPal+col*3+2);
      if (mode==1)
      {
        _PutPixel(Buff,x*2,y*2,r*4,g*4,b*4);
        _PutPixel(Buff,x*2+1,y*2,r*4,g*4,b*4);
        _PutPixel(Buff,x*2,y*2+1,r*4,g*4,b*4);
        _PutPixel(Buff,x*2+1,y*2+1,r*4,g*4,b*4);
      } else
        _PutPixel(Buff,x,y,r*4,g*4,b*4);
    }
  }
  fclose(fpic);
}

int main()
{
  float ir,ig,ib;
  int finish=0;
  ir=100;ig=100;ib=100;
  OpenVGA24BIT(MOD640);
  MaxX=639;MaxY=479;
  buffer=(char *)malloc(MEM640);
  back=(char *)malloc(MEM640);
  sprite=(char *)malloc(200000);
```

Alokasikan memory untuk virtual screen buffer dan back dan juga untuk sprite.

```
  ClearDouble(buffer,0);
```

```
Install_Palette("light.pal");
LoadPicture(0,"light.pic",buffer);
```

Baca file LIGHT.PIC sebagai sprite cahaya dan LIGHT.PAL sebagai susunan palettenya.

```
GetImage(buffer,0,0,319,199,sprite);
```

Rekam sprite.

```
free(devPal);
free(actPal);
Install_Palette("town.pal");
ClearDouble(back,0);
LoadPicture(1,"town.pic",back);
```

Baca file background dan masukkan ke dalam virtual screen back.

```
while(!finish)
{
  MoveDouble(back,buffer);
  Put24BitMask(buffer,160,100,sprite,3,ir,ig,ib);
```

Tampilkan sprite dengan mode efek = 3 (efek cahaya) dan dengan proporsi dominasi unsur ir, ig dan ib.

```
  MoveToScreen(buffer);
```

Tampilkan virtual screen ke layar monitor.

```
  while (!kbhit())
  {
  }
  ch=getch();
  if (ch==27) finish=1;
  if (ch=='q') ir+=5;
  if (ch=='a') ir-=5;
  if (ch=='w') ig+=5;
  if (ch=='s') ig-=5;
  if (ch=='e') ib+=5;
  if (ch=='d') ib-=5;
  if (ch==72)
  {
    ir+=5;ig+=5;ib+=5;
  }
  if (ch==80)
  {
    ir-=5;ig-=5;ib-=5;
```

```
    }
    if (ir<0) ir=0;
    if (ir>255) ir=255;
    if (ig<0) ig=0;
    if (ig>255) ig=255;
    if (ib<0) ib=0;
    if (ib>255) ib=255;
```

Atur ketiga proporsi dominasi sesuai dengan penekanan tombol keyboard.

```
  }
  CloseVGA24BIT();
```

Tutup mode graphic 24 bit.

```
  return 1;
}
```

Sekarang mari kita lihat bagian dari Put24BitMask yang menangani efek cahaya ini.

```
    if (model==3)
    {
      for (a=x1;a<=x2;a++)
      {
        for (b=y1;b<=y2;b++)
        {
```

Lakukan loop untuk memproses seluruh titik pada sprite.

```
          rx=a-x;ry=b-y;
          cx=ry*length*3+rx*3;
          rr=*(Pic+cx+4);
          gg=*(Pic+cx+5);
          bb=*(Pic+cx+6);
```

Ambil intensitas R, G dan B dari titik sprite.

```
          bgb=*(Buff+b*(MaxX+1)*3+a*3);
          bgg=*(Buff+b*(MaxX+1)*3+a*3+1);
          bgr=*(Buff+b*(MaxX+1)*3+a*3+2);
```

Ambil intensitas R, G dan B dari background.

```
          resr=(255.0-bgr)/256.0;
          resg=(255.0-bgg)/256.0;
          resb=(255.0-bgb)/256.0;
```

Hitung pembagian dengan 256 dari sisa antara intensitas maksimum dengan intensitas yang digunakan oleh titik untuk setiap unsur warnanya. Cara ini persis seperti yang telah kita lakukan pada fungsi VESA_Lightning. Karena dilakukannya pembagian dengan 256 maka kita akan memiliki 256 intensitas terang. WOW!!! Pada mode 8 bit kita hanya memiliki 32 intensitas terang. Jelas hasil yang akan kita dapat jauh lebih bagus.

```
rr-=ir;gg-=ig;bb-=ib;
```

Kurangi intensitas titik cahaya dengan parameter. Dengan ini, semakin besar konstanta yang kita kirim untuk sebuah unsur warna, semakin berkurang proporsi dominasi dari warna tersebut dalam pencahayaan.

```
if (rr<0) rr=0;
if (gg<0) gg=0;
if (bb<0) bb=0;
```

Batasi intensitas titik cahaya pada minimum = 0.

```
resr=resr*rr+bgr;
resg=resg*gg+bgg;
resb=resb*bb+bgb;
```

Tambahkan warna titik background dengan cahaya yang didapat dari setiap unsur warna dari titik cahaya.

```
if (resr>255) resr=255;
if (resg>255) resg=255;
if (resb>255) resb=255;
```

Batasi unsur warna pada maksimum = 255.

```
if (rr==0&&gg==0&&bb==0); else
  _PutPixel(Buff,a,b,resr,resg,resb);
```

Tampilkan hasil pencahayaan.

```
        }
      }
    }
  }
}
```

Fungsi transparansi yang dipakai dengan benar pun dapat menghasilkan efek cahaya. Cobalah program TTRANS2.EXE. Di sana akan

ditampilkan sprite yang berwarna-warni dan akan menimbulkan pencahayaan yang sangat bagus. Kalian juga dapat menekan tombol yang sama seperti TTRANS.EXE untuk mengganti proporsi dominasi unsur warna.

Bagi para pembaca yang berminat mempelajari lebih lanjut tentang efek-efek graphic dapat mengikuti buku berjudul PENGOLAHAN GAMBAR SECARA DIGITAL. Di sana dibahas teknik-teknik yang lebih dalam tentang memperbaiki gambar yang rusak, efek pencahayaan, foto editing, efek gelombang air, kaca pembesar dan lain sebagainya.

7.8 Fungsi-fungsi Tambahan

Fungsi-fungsi selanjutnya adalah fungsi-fungsi tambahan yang mungkin dapat berguna bagi para pembaca.

Fungsi Put1Col

Fungsi ini digunakan untuk menampilkan sprite dalam format 8 bit ke virtual screen dalam format 24 bit dengan menggunakan warna yang ditentukan yaitu warna yang diberikan lewat parameter R, G dan B. Fungsi ini mirip seperti fungsi Put1Col pada VGA.ASM.

```
void Put1Col(char * Buff,long x,long y,char * Pic,long
R,long G,long B)
{
```

Parameter yang dikirimkan ialah Buff yaitu alamat virtual screen yang akan digambari. (x,y) ialah koordinat virtual screen di mana sprite akan ditampilkan. Pic ialah alamat memory tempat sprite disimpan dan R, G, B berisi warna yang digunakan untuk menampilkan sprite.

```
  int a,b;
  int x1,y1,x2,y2,skipx,skipy;
  long rx,ry,length,cx;
  char cread;
  skipx=0;skipy=0;
  x1=x;y1=y;
  GetImageDimension(Pic,x2,y2);
```

Ambil panjang dan lebar sprite dengan memanggil fungsi GetImage-Dimension.

```
length=x2;
x2+=x1;y2+=y1;
x2-=1;y2-=1;
```

Hitung koordinat kanan bawah dari sprite yaitu dengan menambahkan koordinat kiri atas sprite dengan panjang dan lebar dari sprite.

```
if (x1<0)
{
  x1=0;
  skipx=-x;
}
if (y1<0)
{
  y1=0;
  skipy=-y;
}
```

Potong sprite terhadap kiri atas layar.

```
if (x2>MaxX)
{
  x2=MaxX;
}
if (y2>MaxY)
{
  y2=MaxY;
}
```

Potong sprite terhadap kanan bawah layar.

```
if (x1<=x2&&y1<=y2&&x1<=MaxX&&y1<=MaxY)
{
```

Jika sprite masih berada di dalam layar, lanjutkan peletakan sprite.

```
  for (a=x1;a<=x2;a++)
  {
    for (b=y1;b<=y2;b++)
    {
```

Lakukan loop untuk memproses seluruh titik pada sprite.

```
      rx=a-x;ry=b-y;
      cx=ry*length+rx;
```

Hitung offset dari sprite dengan informasi rx dan ry yang merupakan indeks horizontal dan indeks vertikal dari titik-titik dalam sprite.

```
        cread=*(Pic+cx+4);
```

Ambil warna titik yang sedang diproses. Dilakukan penambahan dengan 4 karena pada awal sprite, disimpan 4 byte sebagai informasi panjang dan lebar.

```
        if (cread!=0)
          _PutPixel(Buff,a,b,R,G,B);
```

Jika cread tidak sama dengan 0 maka titik harus ditampilkan.

```
      }
    }
  }
}
```

Fungsi Put256

Fungsi ini digunakan untuk meletakkan sprite dalam format 8 bit ke virtual screen dalam format 24 bit. Untuk itu kita harus mengirimkan alamat memory dari tabel palette yang dipergunakan. Sebenarnya cara kerjanya hampir sama dengan LoadPicture pada program-program kita terdahulu dengan memanfaatkan tabel palette untuk mengetahui intensitas R, G dan B dari tiap titik.

```
void Put256(char * Buff,long x,long y,char * Pic,char * PalTab)
{
```

Parameter yang dikirimkan ialah:

```
Buff    = Virtual screen di mana gambar akan diletak-
          kan.
X,Y     = Koordinat tempat sprite akan ditampilkan.
Pic     = Alamat memory tempat sprite disimpan.
PalTab  = Alamat memory tempat tabel palette di- sim-
          pan.
```

```
  int a,b;
  int x1,y1,x2,y2,skipx,skipy;
  long rx,ry,length,cx;
  char cread;
  skipx=0;skipy=0;
  x1=x;y1=y;
```

```
  GetImageDimension(Pic,x2,y2);
  length=x2;
  x2+=x1;y2+=y1;
  x2-=1;y2-=1;
  if (x1<0)
  {
    x1=0;
    skipx=-x;
  }
  if (y1<0)
  {
    y1=0;
    skipy=-y;
  }
  if (x2>MaxX)
  {
    x2=MaxX;
  }
  if (y2>MaxY)
  {
    y2=MaxY;
  }
  if (x1<=x2&&y1<=y2&&x1<=MaxX&&y1<=MaxY)
  {
    for (a=x1;a<=x2;a++)
    {
      for (b=y1;b<=y2;b++)
      {
        rx=a-x;ry=b-y;
        cx=ry*length+rx;
        cread=*(Pic+cx+4);
        if (cread!=0)
          _PutPixel(Buff,a,b,(*(PalTab+cread*3))*4,
          (*(PalTab+cread*3+1))*4,(*(PalTab+cread*3+2))*4);
      }
    }
  }
}
```

Fungsi Convert256to24Bit

Fungsi ini digunakan untuk mengubah sprite dalam format 8 bit ke 24 bit. Fungsi ini menghasilkan hasil output berupa alamat memory dari sprite format 24 bit yang telah diciptakan dan menghapus alokasi memory sprite 8 bit. Sama halnya seperti Put256, fungsi ini mengharapkan dikirimkannya tabel Palette.

```
char * Convert256to24Bit(char * Spr,char * PalTab)
{
```

Parameter yang dikirimkan ialah:

```
Spr      = Sprite dalam format 8 bit yang akan diubah.
PalTab   = Tabel palette yang akan digunakan.
```

```
  int x,y,a,b,ctr=0;
  int cread,cx;
  char * newSpr;
  GetImageDimension(Spr,x,y);
```

Baca panjang dan lebar sprite.

```
  newSpr=(char *)malloc(x*y*3+4);
```

Deklarasikan alamat memory yang baru untuk sprite baru dalam format 24 bit.

```
  memcpy(newSpr,Spr,4);
```

Masukkan ke dalam 4 byte pertama sprite baru.

```
  for (b=0;b<y;b++)
  {
    for (a=0;a<x;a++)
    {
```

Lakukan loop untuk memproses seluruh titik.

```
      cx=b*x+a;
      cread=*(Spr+cx+4);
      *(newSpr+4+ctr)=(*(PalTab+cread*3))*4;
      *(newSpr+4+ctr+1)=(*(PalTab+cread*3+1))*4;
      *(newSpr+4+ctr+2)=(*(PalTab+cread*3+2))*4;
```

Masukkan intensitas R, G dan B dari tiap titik ke sprite baru dalam grup 3 byte.

```
      ctr+=3;
    }
  }
  free(Spr);
```

Hapus sprite lama.

```
  return newSpr;
```

Kembalikan alamat memory sprite baru sebagai hasil fungsi.

}

Dengan ini berakhirlah sudah pembahasan kita mengenai Pemrograman Animasi dan Game Profesional 4 tentang Super VGA 8, 16, 24, 32 bit. Yah... kita bakal berpisah lagi nih. Tenang... sesegera mungkin kita akan bertemu lagi dalam topik-topik yang lebih menarik.

Semoga informasi yang diberikan pada buku ini banyak membantu para pembaca dalam dunia pemrograman animasi dan graphic. Apa yang bakal muncul pada edisi berikutnya? Saya masih belum ada ide. Bagi para pembaca yang ingin memberikan masukan ide dapat mengirimkan surat.

KUNJUNGI ELEX MEDIA KOMPUTINDO online DI CYBERSPACE
http://www.elexmedia.com

INFORMASI GRATIS DARI PENERBIT
BUKU KOMPUTER BERMUTU

Anda akan mendapatkan sesuatu yang bermanfaat setelah mengunjungi kami

- **katalog online** — informasi buku yang tepat untuk Anda
- **daftar buku terbaru** — informasi buku terbaru
- **rencana STO** — informasi produk baru PT Elex Media Komputindo bagi penjual
- **artikel menarik** — tip dan trik menggunakan komputer dan penanganan masalah komputer
- **Mikrodata Online** — media hobiis dan programmer komputer
- **Audiophile Online** — media hobiis audio (diupdate setiap Kamis)
- **Internet Online** — media penjelajah internet (diupdate setiap Sabtu)

Anda dapat bertanya, mengkritik, memuji, memberi pesan dan kesan terhadap produk PT Elex Media Komputindo melalui e-mail kami: **elex@elexmedia.com**

Anda tertarik dengan Internet?
Baca buku-buku Internet terbitan kami

SERI APLIKASI PEMROGRAMAN

PEMROGRAMAN ANIMASI DAN GAME PROFESIONAL 4

SAP Pemrograman Animasi dan Game Profesional 4 merupakan buku seri lanjutan dari seri-seri sebelumnya yang ditulis oleh penulis yang sama. Buku seri keempat ini akan membahas mengenai pemrograman Super VGA dan 32 bit protected mode yang akan membawa kita ke dunia pemrograman graphic resolusi tinggi hingga 1280 x 1024. Dan dibahas pula tahap demi tahap bagaimana menampilkan gambar dan animasi pada mode graphic 8 bit, 16, 24 dan juga 32 bit, yang memungkinkan kita menampilkan gambar dengan kualitas seperti apa yang Anda lihat pada software Adobe Photo Styler. Seiring dengan pembahasan mode graphic tersebut akan dibahas pula teknik-teknik dan efek-efek yang hanya dapat diciptakan dengan mode graphic di atas 8 bit seperti teknik pencahayaan, multi channel transparansi antara dua gambar, pemberian bayangan dan juga efek gamma correction yang ada pada software Adobe Photo Styler. Ingin mengetahui lebih lanjut? Teruskan membaca.

Pembahasan yang Disajikan Mencakup:

- **Berpindah dari Turbo Pascal ke C**
- **Pemrograman dengan Watcon C**
- **Dasar Pemrograman Super VGA**
- **Perakitan Graphics Library 8 Bit**
- **Pemrograman Super VGA 16, 24, 32 Bit**

Tentang Penulis:
Agustinus Nalwan adalah lulusan Computer Science & Rebotic Digital Technology, Monash University Clayton Campus, Melbourne - Australia dan penulis buku-buku komputer khususnya mengenai pemrograman. Hasil karyanya yaitu: *Seri Aplikasi Pemrograman*: **Pemrograman Animasi dan Game Profesional Buku ke 1 - 3, Pengolahan Gambar Secara Digital**, dan akan terus menulis buku-buku lainnya.

Penerbit PT Elex Media Komputindo
Jl. Palmerah Selatan 22, Jakarta 10270
Telp. (021) 5483008, 5490666, 5480888
Ext. 3311 - 3315
Web Page: http://www.elexmedia.co.id

Kelompok
Pemrograman
Ketrampilan
☑ Tingkat Pemula
☑ Tingkat Menengah
☐ Tingkat Mahir
Jenis Buku
☑ Referensi
☑ Tutorial
☐ Latihan

ISBN 979-20-0504-8
12198074